(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－41299
（P2002－41299A）
(43)公開日　平成14年２月８日(2002.2.8)

(51)Int.Cl.[7]	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　9/45		Ｃ０６Ｆ　9/44	５３０Ｐ	５Ｂ０７６
9/44		12/00	５４６Ａ	５Ｂ０８１
	５３０	9/44	３２２Ａ	５Ｂ０８２
12/00	５４６	9/06	６２０Ａ	

審査請求　未請求　請求項の数20　ＯＬ　外国語出願　（全 79 頁）

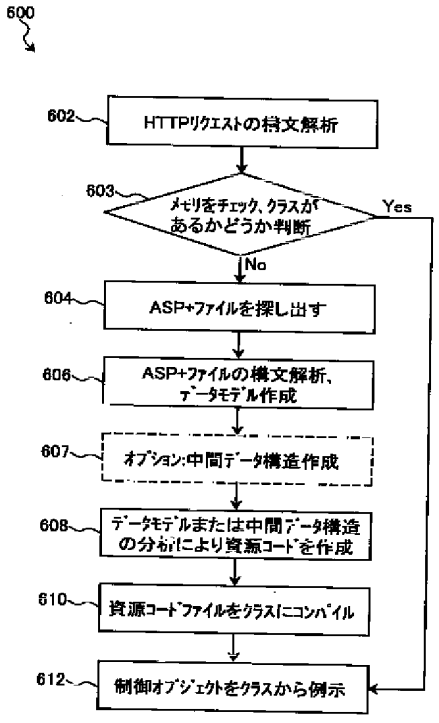
(21)出願番号　特願2001－129924(P2001－129924)
(22)出願日　平成13年４月26日(2001.4.26)
(31)優先権主張番号　０９／５７３７６８
(32)優先日　平成12年５月18日(2000.5.18)
(33)優先権主張国　米国（ＵＳ）

(71)出願人　595124929
マイクロソフト　コーポレイション
ＭＩＣＲＯＳＯＦＴ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　98052－6399　ワシントン州　レドマンド　ワン　マイクロソフト　ウェイ　（番地なし）
(72)発明者　バード、ゲイリー　エス.
アメリカ合衆国、98033　ワシントン州、カークランド、エヌイー　103ド　ストリート　11411
(74)代理人　100095555
弁理士　池内　寛幸　（外３名）

最終頁に続く

(54)【発明の名称】　動的ウェブページコンテンツファイルからのサーバ側コード生成

(57)【要約】　（修正有）
【課題】開発者が最小のプログラミングでウェブページを動的に作成および処理できる開発フレームワークを提供する。
【解決手段】サーバ側資源または動的ウェブページファイルから中間言語または資源コードファイルを作成する方法であり、資源コードは、クライアントレスポンスのレンダリングを含むサーバ側機能を行うウェブページ制御オブジェクトを素早く生成することを可能にする実行可能なクラスにコンパイルされる。コード生成スキームは、イベント処理および特定のオブジェクトへの属性の設定を扱うために、階層で連結された制御オブジェクトを作成する。



【特許請求の範囲】
【請求項１】　メモリを有するサーバコンピュータシステムにおいて、メモリ内に、クライアント側コンピュータシステムに運ばれ、前記クライアントコンピュータシステム上でウェブページとして表示されるウェブページコンテンツを動的にレンダリングするためのサーバ側オブジェクトを作成するために、前記サーバコンピュータシステムによって用いられるクラスを作成する方法であって、
前記クライアントから動的なウェブページコンテンツファイルを特定するリクエストを受信する工程と、
前記動的なウェブページコンテンツファイルを処理して、前記ウェブページコンテンツファイルにおいて宣言された制御オブジェクトを表す資源コードを含む資源コードファイルを生成する工程と、
前記資源コードファイルをコンパイルして、表示するウェブページを生成するウェブページオーサリング言語の作成のために１組の階層オブジェクトが例示され得るクラスを生成する工程とを含むクラスを作成する方法。
【請求項２】　前記動的なウェブページコンテンツファイルが、サーバ側宣言データ格納部である請求項１に記載のクラスを作成する方法。
【請求項３】　前記クラスは、前記サーバコンピュータシステムのキャッシュメモリに格納され、前記動的ウェブページコンテンツファイルを特定する別のリクエストへのレスポンスでオブジェクトを例示するために用いることができる請求項１に記載のクラスを作成する方法。
【請求項４】　前記クラスは、磁気記憶媒体に格納され、前記動的なウェブページコンテンツファイルを特定する別のリクエストへのレスポンスでオブジェクトを例示するために用いることができる請求項１に記載のクラスを作成する方法。
【請求項５】　前記動的なウェブページコンテンツファイルの処理工程において、
前記動的なウェブページコンテンツファイルの一部を、階層的に関連付けられた複数のデータオブジェクトを含んでいるデータモデルへ格納するために、前記動的なウェブページコンテンツファイルの構文を解析する工程と、
第１フェーズ時に、前記データモデルの解析に基づいて宣言情報に関する資源コードを生成する工程と、
前記宣言情報に関する資源コードを前記資源コードファイルへ書き込む工程と、
第２フェーズ時に前記データモデルの解析に基づいて制御オブジェクト情報に関する資源コードを生成する工程と、
第２フェーズ時に前記制御オブジェクト情報に関する資源コードを前記資源コードファイルへ書き込む工程とを含む請求項１に記載のクラスを作成する方法。
【請求項６】　前記方法は、
第３フェーズ時に前記データモデルの解析に基づいてレンダリング情報に関する資源コードを生成する工程と、
第３フェーズ時に前記レンダリング情報に関する資源コードを前記資源コードファイルへ書き込む工程とを含む請求項５に記載のクラスを作成する方法。
【請求項７】　前記第１フェーズが実質的に完了すると前記第２フェーズが生じ、前記第２フェーズが実質的に完了すると前記第３フェーズが生じる請求項６に記載のクラスを作成する方法。
【請求項８】　前記第１フェーズ、前記第２フェーズおよび前記第３フェーズが同時に生じる請求項６に記載のクラスを作成する方法。
【請求項９】　前記動的ウェブページコンテンツファイルの処理工程の前に、前記受信したリクエストに関するクラスがコンパイルされメモリに格納されているかどうか判断する工程と、
前記クラスがコンパイルされメモリに格納されている場合、前記処理工程をスキップし、そうでない場合は、処理工程を行うという工程とをさらに含む請求項１に記載のクラスを作成する方法。
【請求項１０】　メモリを有するコンピュータシステムによって搬送波で具体化され、かつクライアント側コンピュータシステムに運ばれ、前記クライアントコンピュータシステム上でウェブページとして表示されるウェブページコンテンツを動的にレンダリングするためのサーバ側オブジェクトを作成するために、前記サーバコンピュータシステムによって用いられるクラスをメモリ内に作成するコンピュータプロセスを実行処理するコンピュータプログラムをコード化しているコンピュータデータ信号であって、
前記コンピュータプロセスが、
前記クライアントから動的なウェブページコンテンツファイルを特定するリクエストを受信する工程と、
前記ウェブページコンテンツファイルにおいて宣言された制御オブジェクトを表す資源コードを含む資源コードファイルを生成するために前記動的なウェブページコンテンツファイルを処理する工程と、
表示するウェブページを生成するウェブページオーサリング言語を生成するために１組の階層オブジェクトが例示され得るクラスを生成するべく前記資源コードファイルをコンパイルする工程とを含むコンピュータデータ信号。
【請求項１１】　メモリを有するコンピュータシステムによって読み取り可能であり、かつクライアント側コンピュータシステムに運ばれ、前記クライアントコンピュータシステム上でウェブページとして表示され、ウェブページコンテンツを動的にレンダリングするためのサーバ側オブジェクトを作成するために前記サーバコンピュータシステムによって用いられるクラスをメモリ内に作成するコンピュータプロセスを実行処理するコンピュー

タプログラムをコード化しているコンピュータプログラム記憶媒体であって、
前記コンピュータプロセスは、
前記クライアントから動的なウェブページコンテンツファイルを特定するリクエストを受信する工程と、
前記ウェブページコンテンツファイルにおいて宣言された制御オブジェクトを表す資源コードを含む資源コードファイルを生成するために前記動的なウェブページコンテンツファイルを処理する工程と、
表示するウェブページを生成するウェブページオーサリング言語を生成するために１組の階層オブジェクトが例示され得るクラスを生成するべく前記資源コードファイルをコンパイルする工程とを含むコンピュータプログラム記憶媒体。
【請求項１２】　メモリを有するサーバコンピュータシステムにおいて、動的にレンダリングされたウェブページコンテンツを有し、１つ以上のクライアント側コンピュータシステムに運ばれ、前記クライアントコンピュータシステム上でウェブページとして表示される複数のウェブページレスポンスを作成する方法であって、
動的ウェブページコンテンツファイルを識別するリクエストを、前記ウェブページのための前記クライアントコンピュータシステムから受信する工程と、
前記動的なウェブページコンテンツファイルの要素を格納するためのデータモデルを作成する工程と、
前記データモデルの評価に基づき、前記動的なウェブページコンテンツファイルに関する資源コードファイルを生成する工程と、
前記資源コードファイルをコンパイルしてメモリにコンパイルされたクラスを作成する工程と、
前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトを例示することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記動的なウェブページコンテンツを前記クライアントコンピュータシステムに運ぶウェブページレスポンスへレンダリングする工程と、
前記ウェブページレスポンスを前記要求しているクライアントコンピュータシステムへ導く工程と、
動的ウェブページコンテンツファイルを識別する第２のリクエストを、前記ウェブページのために受信する工程と、
当該動的ウェブページコンテンツファイルのコンパイルされたクラスがメモリに存在するかどうかを判断する工程と、
前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトを例示することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記動的ウェブページコンテンツを第２のウェブページレスポンスへレンダリングする工程と、
前記第２のウェブページレスポンスを前記要求しているクライアントコンピュータシステムへ導く工程とを含むウェブページレスポンスを作成する方法。
【請求項１３】　メモリを有するコンピュータシステムによって読み取り可能であり、かつ、動的にレンダリングされたウェブページコンテンツを有し、１つ以上のクライアント側コンピュータシステムに運ばれ、前記クライアントコンピュータシステム上でウェブページとして表示される、複数のウェブページレスポンスを作成するコンピュータプロセスを実行処理するコンピュータプログラムをコード化するコンピュータプログラム記憶媒体であって、
前記コンピュータプログラムは、
動的ウェブページコンテンツファイルを識別するようなリクエストを、前記ウェブページのための前記クライアントコンピュータシステムから受信する工程と、
前記動的なウェブページコンテンツファイルの要素を格納するためのデータモデルを作成する工程と、
前記データモデルの評価に基づき、前記動的なウェブページコンテンツファイルに関する資源コードファイルを生成する工程と、
前記資源コードファイルをコンパイルしてメモリにコンパイルされたクラスを作成する工程と、
前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトを例示することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記動的ウェブページコンテンツを前記クライアントコンピュータシステムに運ぶウェブページレスポンスへレンダリングする工程と、
前記ウェブページレスポンスを前記要求しているクライアント側コンピュータシステムへ導く工程と、
動的ウェブページコンテンツファイルを識別する第２のリクエストを、前記ウェブページのために受信する工程と、
当該動的ウェブページコンテンツファイルのコンパイルされたクラスがメモリに存在するかどうか判断する工程と、
前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトを例示することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記動的ウェブページコンテンツを第２のウェブページレスポンスへレンダリングする工程と、
前記第２のウェブページレスポンスを前記要求している

クライアントコンピュータシステムへ導く工程とを含むコンピュータプログラム記録媒体。

【請求項１４】　メモリを有するコンピュータシステムによって搬送波で具体化され、かつ、動的にレンダリングされたウェブページコンテンツを有し、１つ以上のクライアント側コンピュータシステムに運ばれ、前記クライアントコンピュータシステム上でウェブページとして表示される複数のウェブページレスポンスを作成するコンピュータプロセスを実行処理するコンピュータプログラムをコード化するコンピュータデータ信号であって、
前記コンピュータプログラムは、
動的ウェブページコンテンツファイルを識別するリクエストを、前記ウェブページのための前記クライアントコンピュータシステムから受信する工程と、
前記動的なウェブページコンテンツファイルの要素を格納するためのデータモデルを作成する工程と、
前記データモデルの評価に基づき、前記動的なウェブページコンテンツファイルに関する資源コードファイルを生成する工程と、
前記資源コードファイルをコンパイルしてメモリにコンパイルされたクラスを作成する工程と、
前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトを例示することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記動的ウェブページコンテンツを前記クライアントコンピュータシステムに運ぶウェブページレスポンスへレンダリングする工程と、
前記ウェブページレスポンスを前記要求しているクライアントコンピュータシステムへ導く工程と、
動的ウェブページコンテンツファイルを識別する第２のリクエストを、前記ウェブページのために受信する工程と、
当該動的ウェブページコンテンツファイルのコンパイルされたクラスがメモリに存在するかどうか判断する工程と、
前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトを例示することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記動的ウェブページコンテンツを第２のウェブページレスポンスへレンダリングする工程と、
前記第２のウェブページレスポンスを前記要求しているクライアントコンピュータシステムへ導く工程とを含むコンピュータデータ信号。

【請求項１５】　メモリ内に、クライアント側コンピュータシステムに運ばれ、前記クライアントコンピュータシステム上で処理されるオーサリング言語要素を動的にレンダリングするためのサーバ側オブジェクトを作成するために、サーバ側コンピュータシステムによって用いられるクラスを作成するコンピュータプロセスをコンピュータシステムで実行処理するコンピュータプログラムをコード化するコンピュータプログラムプロダクトであって、
前記コンピュータプログラムは、
動的ウェブページ資源を識別するリクエストを、前記資源のための前記クライアントコンピュータシステムから受信する工程と、
前記資源に関する資源コードファイルを生成するために前記資源を処理する工程と、
前記資源コードファイルをコンパイルして、コンパイルされたクラスをメモリに作成し、前記コンパイルされたクラスのオブジェクトの例示を可能にする工程とを含むコンピュータプログラムプロダクト。

【請求項１６】　メモリ内にクラスを作成するコンピュータプロセスをコンピュータシステムで実行処理するコンピュータプログラムをコード化するコンピュータプログラムプロダクトであって、
前記データモデル作成処理は、
前記資源を論理要素に分離し前記論理要素間の関係を識別するために前記資源の構文解析を行う工程と、
階層データモデルを形成する複数の階層的に関連するデータ構造を作成する工程と、
前記データ構成に前記資源の部分を格納する工程とを含む請求項１５に記載のコンピュータプログラムプロダクト。

【請求項１７】　メモリ内にクラスを作成するコンピュータプロセスをコンピュータシステムで実行処理するコンピュータプログラムをコード化するコンピュータプログラムプロダクトであって、
前記処理工程において、
可変の宣言情報に関連する資源コードを生成するために前記資源の第１の解析を行う工程と、
制御オブジェクト情報に関連する資源コードを生成するために前記資源の第２の解析を行う工程とレンダリング情報に関連する資源コードを生成するために前記資源の第３の解析を行う工程と、
前記資源コードファイルに前記資源コードを格納する工程とを含む請求項１５に記載のコンピュータプログラムプロダクト。

【請求項１８】　メモリ内にクラスを作成するコンピュータプロセスをコンピュータシステムで実行処理するコンピュータプログラムをコード化するコンピュータプログラムプロダクトであって、
前記資源コード生成処理工程において、中間データ構造を生成する工程をさらに含み、前記資源コードが、前記中間データ構造から生成される請求項１６に記載のコンピュータプログラムプロダクト。

【請求項１９】　メモリ内にクラスを作成するコンピュータプロセスをコンピュータシステムで実行処理するコンピュータプログラムをコード化するコンピュータプログラムプロダクトであって、
前記中間データ構造の生成処理工程において、
可変の宣言情報に関連する中間データ構造要素を生成するために前記資源の第１の解析を行う工程と、
制御オブジェクト情報に関連する中間データ構造要素を生成するために前記資源の第２の解析を行う工程とレンダリング情報に関連する中間データ構造要素を生成するために前記資源の第３の解析を行う工程と、
前記中間データ構造要素から資源コードを生成する工程とを含む請求項１８に記載のコンピュータプログラムプロダクト。
【請求項２０】　メモリ内にクラスを作成するコンピュータプロセスをコンピュータシステムで実行処理するコンピュータプログラムをコード化するコンピュータプログラムプロダクトであって、
前記中間データ構造は、複数の資源コード言語ファイルに翻訳され得る包括的な記述であって、少なくとも１つの資源コードファイルは、別の資源コード言語ファイルと異なることを特徴とする請求項２０に記載のコンピュータプログラムプロダクト。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、一般にウェブサーバフレームワークに関し、特にウェブページのクライアント側ユーザインタフェース要素を処理する制御オブジェクトを作成するサーバ側コード作成に関する。
【０００２】
【従来の技術】典型的なウェブブラウザは、クライアントシステムに表示するウェブページの外観および基本動作を定義するウェブサーバからデータを受け取る。典型的な手順としては、ユーザが、ワールドワイドウェブ上の資源のグローバルアドレスであるユニホームリソース（資源）ロケータ（「URL」）を特定し、所望のウェブサイトにアクセスする。URLの一例は、“ HYPERLINK "http://www.microsoft.com/ms.htm" http://www.microsoft.com/ms.htm ”である。このURL例の第１の部分は、通信に用いる所与のプロトコル（例えば“http”）を示している。第２の部分は、資源の所在を示すドメイン名（例えば“ HYPERLINK "http://www.microsoft.com" www.microsoft.com ”）を特定している。第３の部分はドメイン内の資源（例えば“ms.htm”と呼ばれるファイル）を特定している。これに従い、ブラウザは、 HYPERLINK "http://www.microsoft.com" www.microsoft.com ドメイン内のms.htmファイルに関連するデータを取り出すためのURL例に関連するHTTP(ハイパーテキストトランスポートプロトコル)リクエストを生成する。www.microsoft.comサイトをホストしているウェブサーバはHTTPリクエストを受け取り、要求されたウェブページまたは資源をクライアントシステムにHTTPレスポンスで戻し、ブラウザに表示する。
【０００３】上記の例の"ms.htm"ファイルは、静的なHTML（ハイパーテキストマークアップ言語）コードを含んでいるウェブページファイルと対応する。HTMLは、ワールドワイドウェブ上でのドキュメント（例えば、ウェブページ）作成に用いられるプレイン（平文）テキストオーサリング言語である。このようなものであるので、HTMLファイルは、HTMLコードを実際の視覚的な画像または音声コンポーネントへ変換するクライアントブラウザによってウェブサーバから取り出され、ウェブページとして表示される。クライアントコンピュータシステムにおいて、このプロセスは、渡されたHTMLファイルに定義されたウェブページ内容を表示する。HTMLを用いて、開発者が、例えば、ブラウザに表示するフォーマット化されたテキスト、リスト、フォーム、テーブル ハイパーテキストリンク、インライン画像および音声、ならびに背景グラフィックスを特定する。しかし、HTMLファイルは、ウェブページコンテンツの動的な生成を本質的にサポートしていない静的ファイルである。ウェブページコンテンツは、表示のためにクライアントに戻されるHTMLコードである。このような動的処理は、単にクライアントブラウザに所定のコードを送信するのではなく、処理ステップの結果、送信に先立ちHTMLコードを生成するサーバ側アプリケーションに関する。
【０００４】より複雑なクライアント・サーバ間の対話をハンドリングするために、サーバ側アプリケーションプログラムは、動的コンテンツ、たとえば、変化している株価または交通情報を与えるような、より複雑なクライアント・サーバ間の対話をハンドリングするように開発されてきた。サーバ側アプリケーションプログラムは、HTTPリクエストを処理し、クライアントにHTTPレスポンスでクライアントへの送信に適切なHTMLコードを生成する。例えば、サーバ側アプリケーションプログラムは、クライアント側へのHTTPレスポンスにおける送信HTMLコードを動的に生成するために、HTTPリクエストにおいてクライアント側によって提供される照会ストリングまたはウェブベースのフォームからのデータを処理することができる。本質的に、サーバ側アプリケーションは、クライアント側からのリクエストにおける情報に基づきカスタマイズされたHTML型ファイルを生成することができる。この場合、サーバ上に格納された静的なHTMLファイルは存在せず、HTMLファイルは、実行時に動的に作成される。サーバ側アプリケーションプログラムの一例として、メモリ機構への１つ以上のフォーマット化されたテキスト書き込み処理のシーケンスを用いて、HTMLコードを生成する場合がある。その後、得られたテキストは、HTTPレスポンスでクライアントシステムに送信され、そこでブラウザに表示される。

【０００５】サーバ側アプリケーションプログラムの開発は、ウェブページ設計に用いる通常のHTMLコード化に精通しているだけでなく、１つ以上のプログラム言語（例えば、C++, Perl, Visual Basic, またはJscript）を含むプログラムベーシックスに精通していることが要求される複雑な作業である。しかし、残念なことに、ウェブページ設計者は、グラフィックデザイナーまたはエディタであることが多く、人間的な感じを与えるが、プログラミング経験がない場合が多い。したがって、プログラミング経験が少ない人が、サーバ側アプリケーションとそのそれぞれのクライアントとのウェブページインタフェースを開発することができるような、ウェブページファイルを作成するための単純化されたウェブページ開発フレームワークを提供することが必要である。したがって、開発者が最小のプログラミングでウェブページを動的に作成および処理することができる開発フレームワークを提供することが望まれている。

【０００６】

【発明が解決しようとする課題】動的ウェブページ生成のプログラム要件を最小にする１つの手段は、マイクロソフト社によって提供されるアクティブサーバページ（ASP）フレームワークである。ASPフレームワークにより、開発者は、典型的に、Visual BasicまたはJscriptコードおよびその他のHTMLコードを含む“ASP”ウェブページファイルを作成できる。ASPファイルは 種々の機能を実行する宣言またはタグならびにVBスクリプトまたはJスクリプトコードを含む。一般に、実際のプログラミングコードを書き込むより、これらの宣言を書き込むほうが簡単である。

【０００７】処理中、HTTPリクエストは、所望の資源としてASPファイルを特定し、その後、ASPファイルを用いてクライアント側へのHTTPレスポンスでの結果HTMLコードを生成する。さらに、ASPファイルは、所与のアプリケーションプログラミング労力を減らすために、予め開発されたまたは第三者のクライアント側ライブラリコンポーネント（例えば、クライアント側”ACTIVEX”制御）およびデータベースまたは他の第３者アプリケーションを参照することができる。

【０００８】単純化されたASPウェブページファイルは、実行時間でスクリプトエンジンにより解釈され得るスクリプトに変換されなければならない。スクリプトエンジンは、典型的に、所望の結果を達成するために、連続または同期的にASPファイルにおける種々の宣言タイプコマンドを実行する。実行処理可能なファイルとしてコンパイルされ格納されたファイルと比較して、一般に、スクリプトエンジンによって実行されたスクリプトファイルは時間がかかる。これは、スクリプトエンジンが単にファイルを実行するだけではなく、解釈機能を果たさなければならないからである。

【０００９】スクリプトファイルを実行処理可能なファイルにコンパイルする際のある１つの問題は、スクリプトファイルには、おそらく種々の言語の組み合わせがある、またはその可能性があるということである。例えば、スクリプトファイルは、HTMLで書かれたコンポーネントおよびVisual Basicで書かれた他のコンポーネントを含む場合がある。スクリプトエンジンは、実行時間にこれらの要素を解釈するために種々の処理を用いるが、異なる言語コンポーネントを１つの言語、すなわち、１つの資源コードファイルに翻訳するためのコンパイラが存在しない。さらに、現在のサーバ側アプリケーションフレームワークにおいて、サーバ側アプリケーション内でクライアント側ユーザインタフェース要素（例えば、テキストボックス、リストボックス、ボタン、ハイパーリンク、画像、音声など）を動的に管理することを要求されるプログラミングは、依然として高度なプログラミング技術と相当な努力を必要とする。これらのサーバ側プロセスが、より複雑になるにしたがって、スクリプトエンジンは、持続的にこの要求を満たし続けることができなくなる。

【００１０】これらおよび他の理由により、本発明がなされた。

【００１１】

【課題を解決するための手段】本発明は、中間言語または資源コードファイルをサーバ側資源から作成するためのコード生成方法および装置に関するものであり、資源コードファイルは、実行処理可能なクラスにコンパイルされる。実行処理可能なクラスにより、クライアントレスポンスのレンダリングを含むサーバ側機能を実行するウェブページ制御オブジェクトの素早い生成が可能になる。本発明のある実施形態において、コード生成スキームは、イベント処理および特定のオブジェクトへの属性の設定を扱うために階層内で接続された制御オブジェクトを作成することができる。さらに、コード生成方法はまた、テンプレートを用いて宣言されたオブジェクトを接続することもできる。

【００１２】より好ましくは、本発明は、サーバコンピュータシステムメモリにおけるクラスの作成方法に関する。クラスは、ウェブページコンテンツを動的にレンダリングするサーバ側オブジェクトを作成するために、サーバコンピュータシステムによって用いられ、ウェブページコンテンツは、クライアント側コンピュータシステムに送られ、クライアントコンピュータシステム上のウェブページとして表示される。処理において、サーバコンピュータシステムは、ウェブページのためのリクエストをクライアントコンピュータシステムから受信し、このリクエストが動的ウェブページコンテンツファイルを識別する。サーバコンピュータは、動的ウェブページコンテンツファイルの要素を格納するためのデータモデルを作成し、データモデルを評価し、データモデルの評価に基づき動的なウェブページコンテンツファイルに関す

る資源コードファイルを生成する。資源コードが作成されると、資源コードファイルがコンパイルされて、コンパイルされたクラスをメモリに作成する。一般に、プロセスは、サーバコンピュータシステムにクラス参照を戻してサーバコンピュータシステムがクラスを用いることが可能になって終了する。

【００１３】他の好ましい実施形態によると、方法は、サーバコンピュータシステム上のキャッシュメモリにクラスを格納する。一旦、キャッシュメモリに格納されると、多数のサーバ側ページオブジェクトが１つのコンパイルされたクラスから例示され、もともとの資源は再度使用されない。ウェブページに対するリクエストを受信する度に、サーバコンピュータシステムは、その動的ウェブページコンテンツファイルのコンパイルされたクラスがメモリにあるかどうかを判断する。要求されたクラスが、メモリに存在しない場合は、それを作成する。クラスがあれば、サーバコンピュータシステムは、ウェブページコンテンツを動的に生成するために、そのクラスからサーバ側処理オブジェクトを例示する。次に、ウェブページコンテンツはレンダリングされクライアントコンピュータシステムに送られる。

【００１４】本発明のさらに別の実施形態によれば、データモデルを評価する方法ステップは、複数のパスでデータモデルの反復トラバースを伴う。各パス時に資源コードが生成され、そのパス時のデータモデルの評価に基づき資源コードファイルに書き込まれる。データモデルは、階層的に連結されたデータ構造を用いて構築される。

【００１５】本発明によるコンピュータプログラムプロダクトの実施形態は、コンピュータシステムによって読み取り可能であり、サーバコンピュータ上にメモリにコンパイルされたクラスを作成するコンピュータプロセスを実行処理するコンピュータプログラムをコード化するコンピュータプログラム記憶媒体を含む。コンパイルされたクラスは、クライアントコンピュータシステム上に表示される要求されたウェブページに対応するレスポンスをレンダリングするためにサーバ側処理オブジェクトを例示するのに用いられる。本発明によるコンピュータプログラムプロダクトの別の実施形態は、コンピュータシステムによる搬送波によって具体化され、サーバにコンパイルされたクラスを作成するためのコンピュータプログラムをコード化するコンピュータデータ信号を含む。

【００１６】

【発明の実施の形態】本発明の実施の形態は、動的ウェブページコンテンツ資源またはファイルによって定義された特定のウェブページのためのメモリにコンパイルされたクラスを生成する方法に関するものである。コンパイルされたクラスの作成は、ウェブページファイルから資源コードファイルの作成を伴う。次に、資源コードファイルをクラスにコンパイルされる。コンパイルされたクラスが、メモリに存在すると、ページオブジェクトが、表示するクライアントに送り返されるレスポンスをレンダリングするために例示され得る。一般に、ページオブジェクトは、ウェブページ上に表示されるクライアント側ユーザインタフェース要素の処理および生成のためのサーバ側制御オブジェクトを伴う。さらに、サーバ側制御オブジェクトの階層が、これらのオブジェクトが、クライアント上のウェブページの表示のためのHTMLのような結果オーサリング言語コードを最終的に協働して生成するウェブページファイルに宣言され得る。

【００１７】図１は、本発明のある実施形態におけるクライアントに表示するウェブページコンテンツを動的に生成するウェブサーバを示す。クライアント１００は、クライアント１００の表示装置上にウェブページ１０４を表示するブラウザ１０２を実行処理する。クライアント１００は、ビデオモニタ（図示せず）のような表示装置を有するクライアントコンピュータシステムを含む。マイクロソフト社によって販売されている"INTERNET EXPLORER"ブラウザは、本発明のある実施形態におけるブラウザ１０２の一例である。他のブラウザの例として、"NETSCAPE NAVIGATOR" および "MOSAIC"などがあるが、これに限らない。例示したウェブページ１０４には、テキストボックス制御１０６および２つのボタン制御１０８、１１０が組み込まれている。ブラウザ１０２は、HTTPレスポンス１１２でウェブサーバ１１６からHTMLコードを受信し、HTMLコードにより記述されたウェブページを表示する。ある実施形態を参照してHTMLについて説明するが、特に制限されるものではなく、SGML（Standard Generalized Markup Language）およびXML（eXtensible Markup Language）を含む他のオーサリング言語も本発明の範囲内であると考えられている。

【００１８】クライアント１００とウェブサーバ１１６との通信は、HTTPリクエスト１１４およびHTTPレスポンス１１２のシーケンスを用いて行うことができる。ある実施形態を参照してHTTPを説明するが、特に制限されるものではなく、S－HTTPを含めた他のトランスポートプロトコルも本発明の範囲内であると考えられている。ウェブサーバ１１６において、HTTPパイプラインモジュール１１８が、HTTPリクエスト１１４を受け取り、URLを解析し、リクエストを処理する適切なハンドラ１２０を呼び出す。本発明のある実施形態において、異なるタイプの資源を扱う複数のハンドラ１２０がウェブサーバ１１６に備わっている。

【００１９】例えば、URLがHTMLファイルのような静的なコンテンツファイル１２２を特定する場合、ハンドラ１２０は、静的コンテンツファイル１２２にアクセスし、HTTPパイプライン１１８を介して静的コンテンツファイル１２２をHTTPレスポンス１１２でクライアント１００へ送る。あるいは、本発明のある実施形態におい

て、URLがASP+（Active Server Page +）ページのような動的コンテンツ資源またはファイル１２４を特定する場合、ハンドラ１２０は、動的コンテンツファイル１２４にアクセスし、動的コンテンツファイル１２４のコンテンツを処理し、ウェブページ１０４用の結果HTMLコードを生成する。一般に、ファイル１２４のような動的コンテンツ資源は、クライアントに表示すべきウェブページを記述するオーサリング言語を動的に生成するのに用いることができるサーバ側宣言データ格納部である。そして、ウェブページ用のHTMLコードは、HTTPパイプライン１１８を通って、HTTPレスポンス１１２でクライアント１００へ送られる。

【００２０】この処理の間、ハンドラ１２０はまた、開発労力を簡素化するために予め開発された、または第３者コードのライブラリにアクセスすることができる。このようなライブラリの１つがサーバ側クラス制御ライブラリ１２６であり、ここから、ハンドラ１２０は、ユーザインタフェース要素を処理しウェブページに表示する結果HTMLデータを生成するサーバ側制御オブジェクトを例示することができる。本発明のある実施形態において、１つ以上のサーバ側制御オブジェクトは、動的コンテンツファイル１２４に記述されたウェブページ上に、可視的にまたは隠して、１つ以上のユーザインタフェース要素にマッピングする。

【００２１】ハンドラ１２０はまた、ウェブサーバ１１６上または別のアクセス可能ウェブサーバ上で実行処理する１つ以上の非ユーザインタフェースサーバコンポーネント１３０にアクセスする。株価検索アプリケーションまたはデータベースコンポーネントのような非ユーザインタフェースサーバコンポーネント１３０は、ハンドラ１２０によって処理される動的コンテンツファイル１２４において参照され、またはそれに関連付けられる。非ユーザインタフェースサーバコンポーネント１３０は、動的コンテンツファイル１２４で宣言されたサーバ側制御オブジェクトによって立ち上げられたイベントを処理し得る。その結果、サーバ側制御オブジェクトによって提供される処理は、ウェブページのユーザインタフェース要素の処理および生成をカプセル化することによって非ユーザインタフェースサーバコンポーネント１３０のプログラミングを簡単にし、これにより、非ユーザインタフェースサーバコンポーネント１３０の開発者は、ユーザインタフェース問題ではなく、アプリケーション固有の機能の開発に集中することができる。

【００２２】図２は、本発明のある実施形態におけるサーバ側制御オブジェクトを用いてのクライアント側ユーザインタフェース要素の処理および生成処理のフローチャートを示す。処理２００において、クライアントは、HTTPリクエストをサーバに送信する。HTTPリクエストは、ASP+ページなどの資源を特定するURLを含む。処理２０２において、サーバは、HTTPリクエストを受信し、特定された資源を処理する適切なハンドラを呼び出す。ASP+ページは処理２０３において読み出される。処理２０４は、特定された動的コンテンツファイル（例えば、ASP+ページ）の内容に基づきサーバ側制御オブジェクト階層を生成する。

【００２３】処理２０６において、制御オブジェクト階層のサーバ側制御オブジェクトは、ポストバックイベントハンドリング、ポストバックデータハンドリング、状態管理およびデータ結合のうちの１つ以上の処理を行う。処理２０８において、階層内の各サーバ側制御オブジェクトが、クライアント側ユーザインタフェース要素のウェブページでの表示のためのHTMLコードのようなデータを生成（またはレンダリング）するために呼び出される。用語「レンダリング」は、ユーザインタフェース上にグラフィックスを表示する処理を意味することがあるが、本明細書において、用語「レンダリング」は、表示およびクライアント側機能のためのブラウザのようなクライアントアプリケーションによって解釈され得るオーサリング言語データの生成処理をも意味している。処理２０６およびレンダリング処理２０８のより詳細な説明は図６との関連で行われている。個々の制御オブジェクトにおけるrenderメソッドの呼び出しは、ツリートラバーサルシーケンスを用いて行われる。すなわち、ページオブジェクトのrenderメソッドの呼び出しは、階層内の適切なサーバ側制御オブジェクトにわたる反復トラバースになる。

【００２４】あるいは、個々のサーバ側制御オブジェクトの実際の作成は、サーバ側制御オブジェクトが処理２０６または２０８においてアクセスされる（ポストバック入力のハンドリング、状態のロード、制御オブジェクトからHTMLコードのレンダリングなど）まで遅らせてもよい。サーバ側制御オブジェクトが所与のリクエストのためにアクセスされることがない場合、制御オブジェクトの作成を遅らせ、不要な制御オブジェクト作成処理を排除することによって、サーバ処理が最適化される。

【００２５】処理２１０において、HTMLコードをHTTPレスポンスでクライアントに送信する。処理２１４において、クライアントは、表示されるべき新たなウェブページに関連するHTMLコードを受信する。処理２１６において、クライアントシステムは、HTTPレスポンスから受け取ったHTMLコードに応じて新しいページのユーザインタフェース要素を表示する。処理２１２において、サーバ側制御オブジェクト階層を終了する。階層内サーバ側制御オブジェクトは、関連付けられたASP+ページを参照するHTTPリクエストに応答して作成され、オーサリング言語データ（例えば、HTMLデータ）のレンダリングが終わると破壊される。あるいは、処理２１２は、処理２０８の後、処理２１０の前に行ってもよい。

【００２６】図３は、本発明のある実施形態において用いるウェブサーバでのモジュールの一例を示す。ウェブ

サーバ３００は、HTTPパイプライン３０４にHTTPリクエスト３０２を受信する。HTTPパイプライン３０４は、ウェブページ統計のロギング、ユーザ照合、ユーザアクセス権およびウェブページの出力キャッシュ化用モジュールなどの種々のモジュールを含んでもよい。ウェブサーバ３００によって受信される各入力HTTPリクエスト３０２は、最終的には、インタフェースハンドラ（Interface Handler） たとえば、IHTTPハンドラクラス（ハンドラ３０６として図示）の特定のインスタンスによって処理される。ハンドラ３０６は、URLリクエストを解析し適切なハンドラファクトリ（例えば、ページファクトリモジュール３０８）を呼び出す。

【００２７】図３において、ASP+ページ３１０に関連付けられたページファクトリ３０８が呼び出され、ASP+ページ３１０からのオブジェクトの例示および構成をハンドリングする。ASP+ページ３１０は、一意のURLによって認識または参照され得、他の接頭辞も用い得るが、例えば、".aspx"という接頭辞によってさらに識別することができる。特定の".aspx"資源に対するリクエストがまず、ページファクトリモジュール３０８によって受信されると、ページファクトリモジュール３０８は、ファイルシステムを検索して適切な資源またはファイル（例えば、.aspxページ３１０）を得る。このファイルは、リクエストを処理するサーバによって後に解釈またはアクセスされ得るテキスト（例えば、オーサリング言語データ）または別のフォーマットでのデータ（例えば、バイトコードデータまたはコード化されたデータ）を含んでもよい。物理的なファイルが存在する場合、ページファクトリモジュール３０８は、ファイルを開き、そのファイルをメモリに読み出す。あるいは、要求されたファイルは存在するが、予めメモリにロードされている場合は、以下に詳細に述べるように、資源は必ずしもメモリにロードされる必要はないかもしれない。要求されたaspxファイルが見つからない場合、ページファクトリモジュール３０８は、例えばHTTP"404"メッセージをクライアントに送信することによって、適切な「ファイルが見つからない」というエラーメッセージを戻す。

【００２８】ASP+ページ３１０をメモリに読み出した後、ページファクトリモジュール３０８は、ファイル内容を処理し、ページのデータモデル（例えば、スクリプトブロックのリスト、ディレクティブ、静的テキスト領域、階層サーバ側制御オブジェクト、サーバ側制御プロパティなど）を構築する。データモデルは、ページオブジェクトの構造、プロパティおよび機能を定義するコードであるページベースのクラスを拡張させるCOM+（Component Object Model+）クラスのような新たなオブジェクトクラスのソースコードファイルを生成するのに用いられる。本発明のある実施形態において、ソースリストは、中間言語に動的にコンパイルされる。のちにプラットフォームに固有の命令（例えば、X86、Alphaなど）に適時に（Just-In-Time）コンパイルされる。中間言語は、COM+ IL コード、Java バイトコード、Modula 3 コード、SmallTalk コードおよびVisual Basicコードなどの汎用またはカスタム指向言語コードを含んでもよい。別の実施形態において、中間言語処理を省き、固有の命令をソースリストまたはソースファイル（例えば、ASP+リソース３１０）から直接生成する。制御クラスライブラリ３１２は、制御オブジェクト階層の生成に用いられる予め定義されたサーバ側制御クラスを得るためにページファクトリモジュール３０８によってアクセスされ得る。

【００２９】ページファクトリモジュール３０８は、コンパイルされたクラスからページオブジェクトを作成する。ページオブジェクト３１４は、図１のウェブページ１０４に相当するサーバ側制御オブジェクトである。ページオブジェクト３１４およびその子オブジェクト（例えば、テキストボックスオブジェクト３１８、ボタンオブジェクト３２０および別のボタンオブジェクト３２２）が、制御オブジェクト階層３１６の一例である。他の制御オブジェクトの例は、本発明に従い考えることができ、カスタム制御オブジェクトと同様に、特に制限されるものではなく、表１に示すHTML制御に対応するオブジェクト（後述）も含んでいる。ページオブジェクト３１４は、図１のウェブページ１０４に対応する。テキストボックスオブジェクト３１８は、図１のテキストボックス１０６に対応する。同様に、ボタンオブジェクト３２０は、図１の追加ボタン１０８に対応し、ボタンオブジェクト３２２は、図１の削除ボタン１１０に対応する。ページオブジェクト３１４は、サーバ上の他の制御オブジェクトと階層的に関連している。ある実施形態において、ページオブジェクトは、その子オブジェクトを階層的に含んでいるコンテナオブジェクトである。別の実施形態では、依存関係などの他の形態の階層関係を用いることができる。多数レベルの子オブジェクトを有する、より複雑な制御オブジェクト階層において、１つの子オブジェクトが、他の子オブジェクトのコンテナオブジェクトであってもよい。

【００３０】上記の実施形態において、制御オブジェクト階層３１６の制御オブジェクトは、サーバ３００上で作成および実行され、各サーバ側制御オブジェクトは、クライアント上の対応するユーザインタフェース要素と論理的に対応する。サーバ側制御オブジェクトはまた、協働して、HTTPリクエスト３０２からのポストバック入力をハンドリングし、サーバ側制御オブジェクトの状態を管理し、サーバ側データベースとのデータ結合を行い、クライアントでの結果ウェブページの表示に用いるオーサリング言語データ（例えば、HTMLコード）を生成する。結果オーサリング言語データは、サーバ側制御オブジェクト階層３１６から生成（すなわち、レンダリング）され、HTTPレスポンス３２４でクライアントに送信

される。例えば、結果HTML（または他のオーサリング言語）コードは、ACTIVEXタイプの制御またはその他、ブラウザによって処理されたとき、クライアント側ユーザインタフェース要素（例えば、制御ボタン、テキストボックスなど）を生み出すHTML構成を参照することができる。

【００３１】ASP+リソース３１０でなされた宣言によって、サーバ側制御オブジェクトは、非ユーザインタフェースサーバコンポーネント３３０とクライアント側ユーザインタフェース要素との対話のために、１つ以上の非ユーザインタフェースサーバコンポーネント３３０にアクセスすることができる。例えば、ポストバック入力に応答して、サーバ側制御オブジェクトは、サーバ側イベントをそれらのイベント用に登録された非ユーザインタフェースサーバコンポーネントに対し立ち上げることができる。このように、非ユーザインタフェースサーバコンポーネント３３０は、ユーザとの対話を、ユーザインタフェース要素を介して、これらの要素を表示および処理するのに必要なコードをプログラミングすることなく、行うことができる。

【００３２】図４は、本発明のある実施形態における動的コンテンツ資源の一例の内容を示す。例示された実施例において、ファイル４００は、ある動的コンテンツファイルフォーマット（例えば、ASP+）でプレインテキスト宣言を含んでいる。各宣言は、ファイル４００を読み出すページファクトリモジュール３０８に対し命令を与え、クラスを作成し、最終的にHTMLコードまたはその他、クライアントへHTTPレスポンスで送信するオーサリング言語をレンダリングする適切なサーバ側制御オブジェクトを呼び出す。

【００３３】ファイル４００の第１ラインは、以下のフォーマットにおいてデリミッタ”<%@”と” %>”との間にディレクティブを含む：

<%@ directive [attribute=value] %>

ここで、directiveは、特に制限されるものではなく、”page”、”cache”または”import”を含んでもよい。ディレクティブは、バッファリングセマンテックス、セッション状態要件、エラーハンドリングスキーム、スクリプティング言語、トランザクションセマンテックスおよびインポートディレクティブのような特性を決定するために、動的コンテンツファイルを処理するときにページファクトリモジュール３０８によって用いられる。ディレクティブは、ページファイル内のどこに存在してもよい。

【００３４】第２ラインの<html>は、直接的な「書き込み」命令以外、結果HTMLコードをレンダリングするための情報において追加の処理が行われないようにリテラルとして資源コードファイルへ書き込まれる標準HTML開始タグである。HTML構文において、<html>は、HTMLファイルの始まりを示し、これもまたリテラルであるライン２１の終了タグ</html>と対になっている。

【００３５】コード宣言ブロックは、ファイル４００のライン３～１０に存在する。一般に、コード宣言ブロックは、ページオブジェクトおよび制御オブジェクトメンバ変数ならびにサーバ上で実行処理されるメソッドを定義する。以下のフォーマットにおいて：

```
<script runat = "server" [language = "language"][src = "externalfi
le"]>
.....
</script>
```

ここで、言語およびsrcパラメータは任意である。本発明のある実施形態において、コード宣言ブロックは、”サーバ”に設定された値を有する”runat”属性を含む<script>タグを用いて定義される。任意には、”language”属性を内コードの文法を特定するために用いてもよい。デフォルト言語は、ページ全体の言語構成を表すことができるが、コード宣言ブロックの”language”属性により、開発者は、例えば、Jscript およびPERL (Practical Extraction and Report Language)などの同じウェブページ実行内で異なる言語を用いることができる。<script>タグはまた、任意には”src”ファイルを特定することができ、”src”ファイルは、そこからページコンパイラによる処理のための動的コンテンツ資源にコードが挿入される外部のファイルである。開示されている文法が本実施形態において用いられているが、別の実施形態では、本発明の範囲内で異なる文法を用いることができることを理解すべきである。

【００３６】図４において、２つのサブルーチン、AddButton#ClickおよびDeleteButton#Clickがコード宣言ブロック内においてVisual Basicフォーマットで宣言されている。いずれのサブルーチンも２つの入力パラメータ、”Source”および”E”をとり、クライアント側クリックイベントが対応のボタンに検知されると呼び出される。AddButton#Clickサブルーチンにおいて、ユーザ名テキストボックスでのテキストは、単語”Add”に連結され、メッセージのテキストデータメンバにロードされる。DeleteButton#Clickサブルーチンにおいて、ユーザ名テキストボックスでのテキストは、単語”Delete”に連結され、メッセージのテキストデータメンバにロードされる。図４に示していないが、サーバ側制御オブジェクトのメンバ変数は、ファイル４００のコード宣言ブロックで宣言され得る。例えば、Visual Basic文法を用いると、キーワード”DIM”は、サーバ側制御オブジェクトのデータ変数を宣言する。

【００３７】「コードレンダリングブロック」（図示せず）もまた、動的コンテンツ資源に含まれ得る。コード

レンダリングブロックは、ページレンダリング時間で実行処理される任意の量のコードを含むことができる。本発明のある実施形態において、コードレンダリングブロックは、ページレンダリング時に実行処理される１つの「レンダリング」メソッドで実行処理する。コードの他の部分は、そのレンダリングメソッドで実行処理され得る。コードレンダリングブロックは、以下のフォーマット（他のフォーマットも別の実施形態において考えられるが）を満たす。

【００３８】<% InlineCode %>

ここで“InlineCode”は、ページレンダリング時にサーバ上で実行処理する独立言語型コードブロックまたは制御フローブロックを含んでいる。

【００３９】インライン表現もまた、以下のような例示的な文法を用いて表現レンダリングブロックデリミッタ”<%@”と”%>”との間で用いることができる。

【００４０】<%= InlineExpression %>

ここで、“InlineExpression”ブロックに含まれる表現は、“InlineExpression”からの値を宣言内の適切な場所のホールダに書込むページオブジェクトの”Response.Write(InlineExpression)”への呼び出しによって最終的に包含される。例えば、“InlineExpression”は、以下のようにファイル４００に含まれ得る。

【００４１】<font size = ”<%=x%>” > Hi <%=Name%>, you are <%=Age%>! これは、値”x”に格納されるフォントで挨拶およびある人の年齢についての記述を出力する。その人の名前および年齢は、コード宣言ブロック（図示せず）においてストリングとして定義される。結果HTMLコードは、適切な場所に“InlineExpression”の値を含むようにサーバでレンダリングされHTTPレスポンスでクライアントへ送信される。すなわち、以下のようなものである。

【００４２】<font size = ”12” > Hi Bob, you are 35 !

ファイル４００のライン１１において、<body>は、HTMLドキュメントの本文の始まりを規定するための標準HTMLタグである。フィイル４００のライン２０において、終了タグ</body>もまた示されている。本発明のある実施形態において、<body>および</body>のいずれもリテラルである。

【００４３】HTMLフォームブロックの開始タグ<form>が、図４のファイル４００の本文セクション内、ライン１２に見られる。フォームブロックの終了タグ</form>は、HTMLファイル４００のライン１９に見られる。任意のパラメータ”id”もまた、所与の識別子をフォームブロックに関連付けるためにHTML制御タグに含むことができ、これによって、１つのHTMLファイルに多数のフォームブロックが含まれることが可能になる。

【００４４】ファイル４００のライン１８において、「メッセージ」によって識別されたサーバ側ラベルが宣言される。「メッセージ」ラベルは、ウェブページ上にラベルを表示するために、ファイル４００のライン５および８で宣言されたコードで用いられる。

【００４５】フォームブロック内に、図１のユーザインタフェース要素１０６、１０８および１１０に対応する３つのHTML制御タグ例が示されている。第１のユーザインタフェース要素がテキストボックスに相当するファイル４００のライン１３で宣言される。テキストリテラル”User Name”は、テキストボックスの左側に位置するラベルを宣言する。type=”Text”を有する入力タグは、テキストボックスクライアント側ユーザインタフェース要素をレンダリングするサーバ側制御オブジェクトとして”UserName”という識別子を有するテキストボックスサーバ側制御オブジェクトを宣言する。ファイル４００のライン１５および１６は、それぞれ、図１のボタン１０８および１１０として示されるクライアント側ユーザインタフェース要素を宣言する。”OnServerClick”パラメータは、ファイル４００のコード宣言ブロックで宣言された適切なサブルーチンを特定する。そして、ファイル４００での宣言へのレスポンスで生成されたサーバ側ボタン制御オブジェクトが、クライアント側ボタンのHTMLコードおよびボタンクリックイベントを実行する関連のサーバ側コードをレンダリングする。

【００４６】ファイル４００で宣言されたテキストボックスおよびボタンは、HTMLサーバ制御宣言の例である。初期状態では、ASP+リソース内のすべてのHTMLタグはリテラルテキストコンテンツとして取り扱われ、ページ開発者のプログラミングにおいてアクセスすることができない。しかし、ページ開発者は、”server”に設定された値を有する”runat”属性を用いて指定することによって、HTMLタグが構文解析され、アクセス可能サーバ制御宣言として扱われるべきであることを示すことができる。任意には、各サーバ側制御オブジェクトは、対応する制御オブジェクトのプログラム参照を可能にする一意の”id”属性と関連付けることができる。サーバ側制御オブジェクト上のプロパティ引数およびイベント結合もまた、タグ要素上の宣言名／値属性対を用いて特定することができる（例えば、OnServerClickは”MyButton#Click”対に等しい）。

【００４７】本発明のある実施形態において、HTML制御オブジェクトを宣言する一般的な文法は以下のとおりである。

【００４８】

```
<HTMLTag id = ”Optional Name” runat = server>
.....
</HTMLTag>
```

ここで、"Optional Name"は、サーバ側制御オブジェクトの一意の識別子である。サポートされ得るHTMLタグのリストならびに関連文法およびCOM+クラスを表１に示すが、他のHTMLタグも本発明の範囲内で考えることができる。

【００４９】

【表１】

HTML タグ名	例	COM+ クラス
<a>	 My Link 	AnchorButton
<img>	<img id = "MyImage" runat = server>	Image
		Label
	Some contents	Panel
<form>	<form id = "MyForm" runat = server> </form>	FormControl
<select>	<select id = "MyList" runat = server> <option>One</option> <option>Two</option> <option>Three</option> </select>	DropDownList
<input type = file>	<input id = "MyFile" type = file runat = server>	FileInput
<input type = text>	<input id = "MyTextBox" type = text>	TextBox
<input type = password>	<input id = "MyPassword" type = password>	TextBox
<input type = reset>	<input id = "MyReset" type = reset>	Button
<input type = radio>	<input id = "MyRadioButton" type = radio runat = server>	RadioButton
<input type = checkbox>	<input id = "MyCheck" type = checkbox runat = server>	CheckBox
<input type = hidden>	<input id = "MyHidden" type = hidden runat = server>	HiddenField
<input type = image>	<input type = image src = "foo.jpg" runat = server>	ImageButton
<input type = submit>	<input type = submit runat = server>	Button
<input type = button>	<input type = button runat = server>	Button
<button>	<button id = MyButton runat = server>	Button
<textarea>	<textarea id = "MyText" runat = server> This is some sample text </textarea>	TextArea

【００５０】標準HTML制御タグに加えて、本発明のある実施形態によって、開発者は、HTMLタグセットの外側に共通のプログラム機能についてカプセル化する再利用可能コンポーネントを作成することが可能になる。これらのカスタムサーバ側制御オブジェクトは、ページファイル内の宣言タグを用いて特定される。カスタムサーバ側制御オブジェクト宣言は、"server"に設定された値を有する"runat"属性を含む。任意には、カスタム制御オブジェクトのプログラム参照を可能するために、一意の"id"属性が特定される。さらに、タグ要素における属性対である宣言名／値は、サーバ側制御オブジェクトのプロパティ引数およびイベント結合を特定する。インラインテンプレートパラメータもまた、適切な「テンプレート」の接頭文字列である子の要素を親サーバ制御オブジェクトに与えることによってサーバ側制御オブジェクトに結合されることがある。カスタムサーバ側制御オブジェクト宣言のフォーマットは、次のようになる。

【００５１】<servercntrlclassname id="OptionalName" [propertyname="propval"] runat=server/>

ここで、"servercntrlclassname"は、アクセス可能制御クラスであり、"OptionalName"は、サーバ側制御オブジェクトの一意の識別子であり、"propval"は、制御オブジェクトの任意のプロパティ値を表す。

【００５２】別の宣言文法を用いて、XMLタグ接頭語は、以下のフォーマットを用いることにより、ページ内にサーバ側制御オブジェクトを特定するためのより簡潔な表記法を提供するのに用いることができる。

【００５３】<tagprefix:classname id = "OptionalName" runat = server/>

ここで、"tagprefix"は、所与の制御名スペースライブラリと関連し、"classname"は、関連名スペースライブラリでの制御の名前を表す。任意の"propertyvalu

e”もまたサポートされている。

【００５４】図５を参照すると、本発明の実施形態のコンピュータシステムの一例は、プロセッサユニット５０２、システムメモリ５０４およびシステムメモリ５０４を含む種々のシステムコンポーネントをプロセッサユニット５００に接続するシステムバス５０６を含んでいる従来のコンピュータシステム５００という形態の汎用コンピュータ装置を含んでいる。システムバス５０６は、メモリバスまたはメモリコントローラ、種々のバスアーキテクチャを用いるペリフェラルバスおよびローカルバスを含む幾つかのタイプのバス構造のいずれであってもよい。システムメモリは、再生専用メモリ（ROM）５０８およびランダムアクセスメモリ（RAM）５１０を含んでいる。コンピュータシステム５００内の要素間での情報の転送を助ける基本ルーチンを含んでいる基本入力／出力システム５１２（BIOS）は、ROM５０８に格納されている。

【００５５】コンピュータシステム５００は、さらに、ハードディスクの読み出しおよび書き込みを行うハードディスクドライブ５１２、着脱可能な磁気ディスク５１６の読出しおよび書込みを行う磁気ディスクドライブ５１４およびCD ROM、DVDまたは他の光学媒体のような着脱可能な光ディスク５１９の読出しおよび書込みを行う光ディスクドライブ５１８を含んでいる。ハードディスクドライブ５１２、磁気ディスクドライブ５１４および光ディスクドライブ５１８は、それぞれ、ハードディスクドライブインタフェース５２０、磁気ディスクドライブインタフェース５２２および光ディスクドライブインタフェース５２４によってシステムバス５０６接続されている。ドライブおよびその関連コンピュータ読み取り可能媒体が、コンピュータシステム５００のコンピュータ読み取り可能命令、データ構造、プログラムおよび他のデータの不揮発性記憶部を提供している。

【００５６】本明細書に記載の上記環境例では、ハードディスク、着脱可能な磁気ディスク５１６および着脱可能な光ディスク５１９を用いているが、データ保存可能な他のタイプのコンピュータ読み取り可能媒体を上記システム例に用いることができる。上記動作環境例に用いることができるこれらの他のタイプのコンピュータ読み取り可能媒体は、例えば、磁気カセット、フラッシュメモリカード、デジタルビデオディスク、ベルヌイ(Bernoulli)カートリッジ、ランダムアクセスメモリ（RAM）および再生専用メモリ（ROM）などがある。

【００５７】多数のプログラムモジュールが、ハードディスク、磁気ディスク５１６、光ディスク５１９、ROM５０８またはRAM５１０に格納され、これらは、オペレーティングシステム５２６、１つ以上のアプリケーションプログラム５２８、他のプログラムモジュール５３０およびプログラムデータ５３２を含む。ユーザは、コマンドおよび情報をコンピュータシステム５００にキーボード５３４およびマウス５３６または他のポインティング装置などの入力装置によって入力することができる。他の入力装置としては、例えば、マイクロフォン、ジョイステック、ゲームパッド、サテライトディッシュおよびスキャナなどがある。これらおよび他の入力装置は、システムバス５０６に接続されているシリアルポートインタフェース５４０を介して処理装置５０２に接続されていることが多い。しかし、これらの入力装置はまた、パラレルポート、ゲームポートまたはユニバーサルシリアルバス（USB）などの他のインタフェースによって接続されていてもよい。モニタ５４２または他のタイプの表示装置もまた、ビデオアダプタ５４４などのインタフェースを介してシステムバス５０６と接続している。モニタ５４２に加えて、コンピュータシステムは、典型的には、スピーカおよびプリンタなどの他の周辺出力装置（図示せず）を含む。

【００５８】コンピュータシステム５００は、リモートコンピュータ５４６のような１つ以上のリモートコンピュータへの論理接続を用いたネットワーク化された環境で動作することができる。リモートコンピュータ５４６は、コンピュータシステム、サーバ、ルータ、ネットワークPC、ピア（peer）装置、または他の共通ネットワークノードであり得、典型的にコンピュータシステム５００との関連で上述した要素の多くまたはすべてを含む。ネットワーク接続は、ローカルエリアネットワーク（LAN）５４８およびワイドエリアネットワーク（WAN）５５０を含む。このようなネットワーク環境は、オフィス、企業規模コンピュータネットワーク、イントラネットおよびインターネットにおいて珍しいものではない。

【００５９】LANネットワーク環境で用いるとき、コンピュータシステム５００は、ネットワークインタフェースまたはアダプタ５５２を介してローカルネットワーク５４８に接続される。WANネットワーク環境で用いるとき、コンピュータシステム５００は、典型的に、インターネットのようなワイドエリアネットワーク５５０による通信を確立するためのモデム５５４または他の手段を含む。モデム５５４は内蔵または外付けのいずれでもよく、シリアルポートインタフェース５４０を介してシステムバス５０６と接続されている。ネットワーク化された環境において、コンピュータシステム５００に関連して述べたプログラムモジュールまたはその一部は、リモートメモリ記憶装置に記憶されてもよい。図示されたネットワーク接続は例であって、コンピュータ間の通信リンク確立のために他の手段を用いることができる。

【００６０】本発明の実施形態において、コンピュータ５００は、ウェブサーバを表し、CPU５０２が、記憶媒体５１６、５１２、５１４、５１８、５１９またはメモリ５０４のうち少なくとも１つに記憶されたASP+ファイル上でページファクトリモジュールを実行処理する。HTTPレスポンスおよびリクエストは、クライアントコンピ

ュータ５４６に接続されたLAN５４８により通信される。

【００６１】ページファクトリモジュール３０８によって行われる処理６００を図６のフローチャートに示す。図６では、処理６００は、制御オブジェクト階層２０４の生成処理（図２）にほぼ対応する。処理６００は、ASP+ページ（ASP＋ファイルともいう）をコンパイルされたオブジェクトコードクラスに変換し、ついで、それは、制御オブジェクト３１４、３１８、３２０および３２２（図３）を例示するためにメモリにロードされる。

【００６２】サーバが処理２０２においてURLに対するリクエストを受信すると（図２）、ページ作成処理６００が始まる。リクエストの受信後 構文解析処理６０２が、要求されたURLの構文解析を行い、どの資源が要求されているかを判断する。資源は、静的なファイル（.htm、.gif、.jpgなど）、ディレクトリブラウジング起動、DAV（デジタルオーディオ／ビデオ）ファイルおよび動的なコンテンツリクエスト（.aspx、.soapなど）を含み、本発明は、これらをハンドリングするように構成されている。サーバ側によって受信された、入ってくるHTTPリクエストは、ハンドラクラス、例えば、IHTTPHandlerクラスの特定のインスタンスによって最終的に処理される。ハンドラファクトリ、例えば、IHTTPHandler“ファクトリ”の使用によって、ハンドラインスタンス、例えば、IHTTPHandlerインスタンスへのURLリクエストの解釈を実際に行う差込可能なアーキテクチャが提供される。この解釈は、入ってくるURLのファイル拡張およびHTTPコマンドを、適切なハンドラインスタンス、例えばIHTTPHandlerインスタンスを最終的に作成することになっているIHTTPHandlerFactoryクラスのようなファクトリクラスへ写像することができるアプリケーション構成設定を用いて容易に行うことができる。構文解析処理において、差込可能アーキテクチャにより種々の資源を識別できるが、以下の記載は、動的ウェブページコンテンツ資源を識別するHTTPリクエストに焦点をあて、特に、.aspxまたはASP+ページもしくはファイルのような動的ウェブページコンテンツファイルに焦点を当てている。

【００６３】構文解析処理６０２で実際の資源が識別されると、チェック処理６０３において、メモリをチェックし、クラスがメモリに存在するかどうかを判断する。クラスがコンパイルされているかどうかを判断するために、チェック処理６０３において、クラスがコンパイルされるとセットされるフラグを検索することによって、または名前によってメモリ内のクラスを検索する。どちらにしても、チェック処理６０３において、コンパイルクラスがすでにコンパイルされ格納されていることを示す指標を検索する。クラスがメモリにある場合、フローはYES枝に分岐し例示処理６１２に進む。例示処理は、コンパイルされたクラスから制御オブジェクトを例示する。チェック処理６０３において、クラスがコンパイルされていないと判断されると、フローはNO枝に分岐し位置決定処理６０４に進む。

【００６４】チェック処理６０３が、クラスはメモリに存在しないと判断すると、位置決定処理６０４において、特定の資源を検索し探し当て、ファイルをメモリに読み出す。”System. ASP WebForms. PageFactory”クラスのようなベースクラスが、ASPおよびASPXページの例示および構成をハンドリングするハンドラファクトリインプリメンテーションを提供する。資源、この場合は物理的な動的なウェブページコンテンツファイルが見つからなければ、適切なエラーメッセージが戻される。

【００６５】位置決定処理６０４の次に、処理６００において、構文解析作成処理６０６を行い、ここでASP+ファイルのような資源の構文解析を行う。構文解析／作成処理６０６は、ASP+ファイルを宣言ごとに読み出し、構文解析しながら集めた情報からデータモデルを作成する。データモデルは、資源コードを引き出すことができるアクティブコンテンツファイルの要素に関連する要素を含むデータ構造である。データモデルは、アクティブコンテンツファイルで参照された構造的要素を含み、これらの要素は、アクティブサーバページの結果制御オブジェクトの構造を表すように連結される。本発明のある実施形態において、データモデルは、階層的なツリー構造で関連付けられたオブジェクトの組み合わせである。

【００６６】ウェブコンテンツファイルの各宣言は、データモデルの作成方法およびデータモデルに格納される情報を示す所定の要素を有している。例えば、宣言がリテラルテキスト宣言である場合、データモデルは、テキストが適切な位置に挿入されるような情報を含むだけでよい。しかし、宣言が入れ子の制御オブジェクトが存在することを示す場合、ASP＋ファイルに宣言されたような入れ子に従ったデータモデルが作成されなければならない。本質的に、データモデルは、各宣言ごとに１つのオブジェクトを含み、各オブジェクトは、各オブジェクトに子オブジェクトが存在するかどうかに関する情報を含む。

【００６７】ASP＋ファイルが構文解析され、データモデルが作成されると、任意である作成処理６０７を行ってもよい。作成処理６０７は、以下に記載するように、データモデルを解析し、例えば、作成すべき資源コードファイルのような資源コードを抽象化したものである中間データ構造を作成する。本質的に、中間データ構造は、コードを記述する包括的なデータ構造である。中間データ構造は、クラスを記述する上位レベルのデータ構造を有する。各クラスごとに、クラス名およびメソッドのリストまたはアレイを有する別のレベルのデータ構造であってもよい。さらに、各メソッド自体、ステートメントおよび宣言のような種々の要素を有するデータ構造である。ステートメントは、表現、データ呼び出しなど

のような項目を含み得る。

【００６８】ASP+ファイルは構文解析され、データモデル（および任意には、中間データ構造）が作成されると、作成資源コードモジュール６０８は、典型的には、ASP+ファイルに存在する各宣言のコードの種々のラインを書き込むことにより、各データモデルの解析を通して必要な資源コードを作成する。一般に、このステップは、データモデルにおける公知の宣言および他の情報のコードへの直接的な翻訳を伴う。このような翻訳は、翻訳を行うアプリケーションへハードコード化されるか、または、各翻訳は、検索テーブルに格納され得る。この種の検索テーブルは、ほとんどすべての言語にとって適切な翻訳を提供するよう作成され得るが、オブジェクト指向資源コード言語の使用により、既存のCOMまたはCOM+ライブラリを利用するためのCOMまたはCOM+クラスへのコンパイルが簡単になる。（例えば、資源コード言語の一例として、C++またはVbasicなどがある。）さらに、中間データ構造が処理６０７で作成される場合は、翻訳はさらに直接的になる。このような場合、データ構造は、包括的な資源コード言語ファイルに非常に近くなって、翻訳は、単に、特定の資源コード言語の資源コードファイルを作成するのに適切な辞典および文法の追加になるにすぎない場合がある。

【００６９】ASP+ファイルは、結果資源コードファイルに特定の資源コード言語を宣言する。ASP+ファイルに言語が宣言されていない場合、Visual Basicなどのデフォルト言語を用いることができる。データモデルまたは中間データ構造から生成された結果資源コードファイルは、本質的に、プログラマがサーバ側制御オブジェクト階層を利用する場合で、このコード生成ツールの恩恵を受けないときに書き込むことを要求されることになるタイプの高度な資源コードファイルである。

【００７０】資源コードファイルが完成すると、コンパイル処理６１４において、資源コードファイルをコンパイルし、オブジェクトコードまたは他の実行処理可能フォームでコンパイルされたクラスを作成する。あるいは、クラスは、バイトコードまたは仮想装置上で起動するできる他の言語にコンパイルすることができる。資源コードファイルのコンパイルは、資源コードファイルに存在する資源コード言語をコンパイルするように構成されたコンパイラの使用を伴う。そして、クラスが、例示処理６１６によって呼び出され、ウェブページのためのページオブジェクトおよび結果制御オブジェクトを作成する。URLリクエストに明示的に備わっていないが、例示処理は、特定の資源のURLリクエストの一部である非明示的処理である。

【００７１】ASP+ファイルがページクラスにコンパイルされると、クラスは、将来のリクエストで使用可能な状態で維持される。したがって、ASP+ファイルの資源コードの生成処理は、１度行われるだけでよく、後のリクエストはコンパイルされたクラスを利用することができ、必要なオブジェクトを必要に応じて例示する。コンパイルされたクラスは、短期間の間キャッシュされ得、かつ／またはコンパイルされたクラスはディスクに格納し得る。実際、ASP+ファイルが一旦コンパイルされると、元々のASP+ファイルを再び触れる必要がない。もちろん、ASP+ファイルが修正される場合は、コンパイル処理を繰り返すことによってページクラスを更新する。適切なフラグまたは他の指標メカニズムを用いてASP+ファイルが更新されたかどうかを判断することができ、したがって自動クラス更新が可能である。あるいは、ウェブページファイルまでASP+を更新した後、キャッシュされたクラスを除去するという責任は開発者に残され得る。

【００７２】データモデルの作成に関連した詳細を図７に示す。データモデル作成処理７００は、テスト処理７０２でASP+ファイルにおいて宣言があるかどうかを判断することから始まる。純粋なHTMLファイルのようなコンパイルすべき宣言がない場合、フローは、NO枝に分岐し処理７１４の終了に進む。

【００７３】テスト処理７０２でASP+ファイルに少なくとも１つの宣言があると判断されると、フローはYES枝に分岐し、ASP+ファイルで第１の宣言を得る獲得処理７０４に進む。第１の宣言からの情報は、格納処理７０６によってデータ構造に格納される。データ構造に格納された実際の宣言のみならず、宣言が子オブジェクトを有するコンテナタイプの制御を宣言するかどうか、または宣言がディレクティブなどを宣言するかどうかなどの第１の宣言に関連した情報もデータ構造に格納され得る。ウェブページは、ASP+ファイルであるので、典型的に第１の宣言としてディレクティブタグを有する。ディレクティブタグは、種々のディレクティブ、例えば図４のライン１に示すようなデリミッタ”<%@”と”%>”との間の情報を含む。図４に示すデリミッタは、単に例示的であり他の選択も可能であることを理解すべきである。ディレクティブは、資源コードファイル作成に用いられるページファクトリモジュールに情報を供給する。例えば、図４のライン１に示すディレクティブは、用いられるべきデフォルト言語が、この例では、VB（すなわち、Visual Basic）であることを示している。

【００７４】第１の宣言が取り出されデータ構造に格納されると、処理７００は、ファイルに別の宣言があるかどうかを判断するテスト処理７０８に進む。この処理は、上記の処理７０２と同様である。もはや宣言がなければ、フローはNO枝に分岐し終了ステップ７１４に進む。別の宣言があると、フローはYES枝に分岐し、獲得処理７１０に進み 次の宣言を獲得する。

【００７５】次の宣言の取り出しの次に、格納処理７０８は、データモデルの他のデータ構造に関連付けられた異なるデータ構造に次の宣言に関連する情報を格納する。第１の宣言に関して上で述べたように、データ構造

に格納された情報は、次の宣言のリテラルテクストのみならず、リテラルテキストから派生した情報をも含み得る。したがって、データ構造は、必要に応じて階層的に第１のデータ構造に関連付けられ得る。

【００７６】次のデータ構造への情報の格納の次に、フローはテスト処理７０８に分岐し、資源ファイル、例えば、ASP+ファイルにデータモデルに読み出すべき宣言がまだ存在するかどうかを判断する。存在しない場合は、フローは、NO枝に分岐し処理の終了に進む。しかし。テスト処理７０８がまだ宣言が存在すると判断すると、フローはYES枝に分岐し、次の宣言を獲得する獲得処理７１０に進む。獲得処理７１０の次に、格納処理７０８において、上記のように、次の宣言に関連した情報を格納する。これらのステップ７０８、７１０および７１２は、すべての宣言が順次取り出され、データ構造に格納されるまで続く。データ構造は、入れ子のオブジェクト、子オブジェクト、ノードまたは親オブジェクトのサブ要素は、それらがサブ要素として識別できるように連結される階層として関連付けられてもよい。

【００７７】資源コードモジュール６０８（図６）の作成には、このデータモデルを用いて、特定の資源コード言語の資源コードファイルを作成する。データモデルからの資源コードファイルの作成処理は、資源コード言語に依存することに注目することが重要である。すなわち、特定の言語で書き込まれた資源コードファイルは、典型的に、特定のフォーマット要件、例えば、すべての可変宣言がこれらの変数の使用に先立つという要件を有する。したがって、その言語のコンパイラが呼び出されると、適切なプログラム命令が資源コードファイル内の適切な位置に配置されなければならない。この作業をなし遂げるために、ASP+ファイルを評価して、ファイルの中にどんな一意の要素または特徴があるかを判断しなければならないし、またファイルを処理して資源コードファイルを生成しなければならない。資源コードファイルは、典型的に、第１の一組の要素または宣言が可変宣言のようなファイルの始めにあることを要求するので、データモデル全体が、この情報のために解析されるべきである。同様に、制御オブジェクト情報は、ファイルの中央にあるべきなので、処理の第１フェーズの次の第２フェーズ時にこの情報のためにデータモデル全体を解析すべきである。

【００７８】これらのフェーズは、資源コードファイル（または処理６０７での中間データ構造）に書き込まれるべきコードの別個の部分を判断する機能的処理であり、ここでは、別個の部分は、資源コードファイル内の結果位置に基づき論理的に組み合わされる。したがって、これらのフェーズは、１つのデータモデルの「パス」またはトラバースであると考えられ、したがって連続的に行われる。しかし、あるいは、これらのフェーズは、別個の並行した処理として行われてもよく、別個の処理の結果は１つの資源コードにまとめられ、処理が単一データモデルまたはデータモデルの別個のコピーのいずれかを評価する。あるいは、コードの別個の部分を判断し書き込む機能的な処理またはフェーズは、単一のトラバース時に行われてもよく、そこで、資源コードは、資源コードファイルの別個の部分に選択的に書き込まれたり、その部分が、資源コードの論理的部分間の適切な連結を提供するようにまとめられるか、関連付けられた別個のファイルに書き込まれたりする。したがって、本明細書中では、別個の解析を、実質的に連続して、すなわち次々に行われるように記載しているが、解析は実質的に同時にスプリットされたり、実行されてもよい。

【００７９】図８に示す処理またはフローは、データモデルから比較的直接資源コードファイルを生成する実施形態、すなわち図６に示す省略処理６０７に関する。本実施形態は、変数が始めに宣言され、オブジェクトおよびメソッドが変数情報に続くVBコードファイルのような資源コードファイルを生成する。図８に示す資源コード作成処理８００は、データモデルの「パス」として別個のフェーズを示し、データモデルのデータ構造を３回の通過させる。第１のパスの処理において、変数および宣言情報を探している。第２のパスの処理において、オブジェクト作成情報を探し、第３のパス処理８００において、コードレンダリング情報を探している。

【００８０】第１のパスは、変数および他の宣言情報についてデータモデルをトラバースする処理８０２で始まる。このステップは、処理７００で作成されたデータモデルの第１のデータ構造を解析することから始まる。第１のデータ構造の解析処理において、宣言に関する資源コードファイルに書き込まれるべき資源コードの特定ラインがあるかどうか判断する。

【００８１】第１のデータ構造の解析が終了すると、処理８０４において、資源コードファイルに変数および宣言情報に関する資源コードを生成し書き込む。ライタオブジェクトが呼び出され、それに与えられたテキストを資源コードファイルに単に書き込む。したがって、ライタオブジェクトへの呼び出しは、資源コードファイル名のパラメタと変数または宣言情報に関する特定のプログラミング言語の文法のテキストストリングとの両方を含む。あるいは、テキストは、コピーされるか、または他の公知の手段を用いて資源コードファイルに書き込まれる。

【００８２】また、処理８０４において、ライタオブジェクトへ渡すべき適切な文法を決定しなければならない。本質的に、現在のデータ構造が、サーバ側制御を表す場合は、処理８０４において、それに対する資源コードを生成する。そうでなければ、処理８０４において、それを資源コードファイルにコピーする。本質的に、処理８０４は、３つのうちの１つを行い得る。生成処理８０４において、まず、ASP+ファイルにおける情報がリテ

ラルテキストであるかどうか、したがって資源コードファイルに直接書き込まれるべきかどうかを判断してもよい。これは、HTMLタグがASP+ファイルに挿入される場合であり、この場合、資源コード宣言は生成されない。その代わりに、情報は単に資源ファイルにコピーされる。処理８０４において、第２に、テキストが単純な翻訳、特定の資源コード言語の適切な文法フォームで１対１の対応を要求するかどうかを判断してもよい。このような処理８０４において、１対１対応を探すか、または、ページファクトリモジュールにハードコードされた場合は１対１翻訳を行い、結果情報をライタオブジェクトに供給する。第３に、１対１翻訳ができない場合は、処理８０４において、より複雑な翻訳が要求され、したがってモジュールまたは次の検索テーブル処理を呼び出し、ライタオブジェクトに渡されるべき適切な資源コード文法の作成処理を行う。典型的に、宣言情報の翻訳は直接的である。

【００８３】処理８０４において、第１のデータ構造の資源コードの生成および書き込みを行うと、テスト処理８０６において、データモデルに解析すべきデータ構造がまだ存在するかどうかを検知する。データモデルのこの第１のパス時に解析すべきデータ構造がまだある場合は、フローはYES枝に分岐して処理８００の始めに戻り、８０２において次のデータ構造に対してトラバースを行う。このような場合、次のデータ構造は、上記のように解析され資源コード文法を生成し、その文法コードを資源ファイルに書き込むライタオブジェクトを呼び出す。

【００８４】テスト処理８０６において、この第１のパス時にデータモデルに解析すべきデータ構造がもう存在しないと判断した場合は、フローはNO枝に分岐してトラバース処理８０８に進み、データ構造を通過する第２のパスを開始する。トラバース処理８０８は、典型的にオブジェクト製作およびメソッド作成を書き込むコードを書き込むという次のフェーズを開始する。トラバース処理８０８は、データモデルの上から始まり、データモデルの始めのデータ構造に戻る。

【００８５】次に、生成および書き込み資源コード処理８１０において第１のデータ構造を解析し、オブジェクト作成に関する情報がその第１のデータ構造にあるかどうかを判断する。このようなオブジェクト作成情報が第１のデータ構造に存在する場合、特定のプログラミング言語の文法が作成および生成されて資源コードファイルに書き込むライタオブジェクトに送信される。本質的に、処理８１０は、上記の処理８０４と同様であり、ここでは、データ構造が特定タイプの情報について解析され、その情報がデータ構造から収集されると、何らかの翻訳を行い、その情報に関する資源コード言語を作成する。情報がリテラルである場合、処理８１０において、その情報を資源コードファイルに直接送ることができる。同様に、その情報が簡単な翻訳を必要とする場合は、その簡単な翻訳は、プログラムにハードコードされるか、あるいは検索テーブルタイプ処理の一部であってもよい。また、さらに複雑な構造が要求される場合は、検索テーブルまたは他のモジュールを呼び出し、情報を処理し、資源コードファイルに適切な文法資源コードを作成する。

【００８６】処理８１０の次に、照会ステップ８１２は、データモデルに解析すべきデータ構造が存在するかどうかをテストする。オブジェクト作成のために解析すべきデータ構造がデータモデルに存在する場合、フローはYES枝に分岐し次のデータ構造をトラバースするトラバース処理８０８に進む。したがって、処理８０８において、次のデータ構造に戻り、処理８１０において、オブジェクト作成に関する情報のデータ構造を解析する。このオブジェクト作成情報は、上記のように、ライタオブジェクトに送信され、資源コードファイルに追加することができる資源コードラインに翻訳される。ステップ８１２、８０８および８１０は、すべての資源コードがオブジェクト作成に関し書き込まれるまで繰り返される。

【００８７】判断ステップ８１２が、オブジェクト作成のために解析すべきデータ構造がデータモデルにもう存在しないと判断すると、フローはNO枝に分岐し処理８１４に進み、第２のパスを終了する。この第３のパスは、データモデルのトップの処理８１４で始まる。処理８０８および８０２に関して上で述べたように、このトラバースステップは、まず、データモデル内の第１のデータ構造を取り出す。処理８１４でのデータモデル内の第１のデータ構造の取り出しの次に、処理８１６で、コードレンダリング情報のデータ構造を解析する。コードレンダリング情報は、ライタに送信されたり、資源コードファイルに追加させるプログラミング言語での特定な文法に翻訳される。コードレンダリングメソッドは、資源コードファイルの終わり近くにあり、したがって、第３のパスがオブジェクト作成パスの次に行われる。あるいは、コードレンダリングメソッドが同時スレッド処理に間に生成され、資源ファイルの終わりに追加される。

【００８８】処理８１６は、上記の処理８１０および８０４とかなり似ており、この資源コード情報は、データ構造にあるコードレンダリング情報から収集され得る。さらに、上記のように、情報は直接挿入されてもよく、１対１の割合で翻訳されてもいいし、より複雑なモジュールを用いて資源コードファイルの文法を生成することを要求してもよい。処理８１６の次に、検知処理８１８において、データモデルにデータ構造が残っていないかどうか判断する。

【００８９】処理８１８において、この第３のパス間に解析すべきデータ構造がまだ存在すると検知すると、フローはYES枝に分岐し、次のデータ構造のトラバースの

ために８１４に戻る。次のデータ構造が呼び出され、処理８１８において、次のデータ構造のコードレンダリング情報に関する資源コードの生成および書き込みを行う。

【００９０】除り８１８において、解析すべきデータ構造がもはや存在しないと判断すると、フローはNO枝に分岐し、処理連結８２０で処理を終了する。

【００９１】上記の説明により明らかなように、数回のパスがデータモデルに対して行われる。これは、データモデルの各宣言が、資源コードファイルの種々の場所に置かれるコードの特定のラインを要求することによる。しかし、資源コードファイルを書き込む際、ライタオブジェクトは、ファイルに連続的に追加して順次書き込むことしかできず、予め書き込まれたコードライン間に資源コードラインを挿入できない。したがって、データモデルは、資源コードの第１または上部、資源コードの第２または中間部、および資源コードファイルの第３または最後の部分に属する適切な情報を収集するために１回以上トラバースしなければならない。パスの数は、種々のコンピュータ言語に対し存在するコードの部分の数によって異なることが予想される。

【００９２】図９に示す処理またはフローは、データモデルから中間データ構造を生成し、次いで、中間データ構造から資源コードファイルを生成する実施形態、すなわち図６の処理６０７および６０８の別の実施形態に関する。本実施形態のフロー９００は、処理８０４、８１０および８１６が特定の言語の資源コードを生成し、処理９０４、９１０および９１６が、資源コードを生成するのではなくて、中間データ構造の部分を生成することを除いて図８のフローと同様である。したがって、処理９０４、９１０および９１６が、データモデルでの情報に関する情報を生成する際に処理８０４、８１０および８１６と同様の処理を行う。しかし、生成された情報は包括的であり、後に異なる資源コード言語の資源コードファイルを作成するのに用いられることがある。

【００９３】データモデルに解析すべきデータ構造がもはや存在しないと判断されると、中間データ構造が完成し、フローは分岐して作成処理９２０に進む。作成処理において、中間データ構造から資源コードファイルを作成する。データ構造は資源コードファイルの包括的な記述であるので、処理９２０において中間データ構造を包括的な記述から特定の言語コードファイルに翻訳する。

【００９４】図１０は、ウェブページ開発者によって製作されたウェブページ資源ファイル、すなわちASP+ファイルの例である。ファイルは、資源コードファイルを生成するための処理６００のサブジェクトである。図１０に示すASP+ページは、ライン１にディレクティブライン、ライン３から１３にサーバ側スクリプトブロックおよびライン１６から２１にサーバ側コントロール宣言ブロックを有する。ディレクティブラインは、どのようなタイプの資源コード言語を用い、それに一般的な記述を与える（そうでなければ資源コードファイルのどのコードにもならない）かを判断するために処理６００において用いられる。コード宣言ブロック”<script runat=server>　　</script>”内に書かれたすべてのコードは、概念的にページメンバ宣言（変数、プロパティ、メソッド）として扱われ、図１１に示すような生成ファイルの資源ファイルに直接挿入される。

【００９５】図１１は、図１０に示すASP+ファイルを用いて資源コードを作成する処理によって作成された資源コードファイルの一例である。図１１の第１のラインは、資源コードファイルによって作成しようとしているページクラスは、コンパイルされたとき、System.ASP.WebForms.Pageから引き継いでいることを示している。このクラスからの引継ぎは、ページクラスの多くの制御機能を提供するので重要なステップである。初期状態として、生成された資源ファイルは“System.ASP.WebForms.Page”ベースクラスのサブクラスである。任意には、開発者は、ディレクティブラインに与えられた”Inherits”属性を用いて代替クラスを特定することができる。

【００９６】ライン３および４は、第１のパスでデータモデルを通っている間に作成された可変宣言を示している。

【００９７】第２のパス時にライン６に示された情報が作成され、新しい制御オブジェクト”DataList”を作成し、それをMyDataと名づける。ある実施形態において、ASP+ファイル宣言ブロック”<script runat=server>　</script>”内に書き込まれているすべてのコードは、ページメンバ宣言（変数、プロパティ、メソッド）として扱われ、上記のような生成ファイルの資源ファイルに直接挿入される。このように、コード８－１３のラインは、リテラルテキストとして直接挿入された。データモデル６０６の作成の間、リテラルテキストに関連する情報は、多数のパスの間トラバースされないように、本質的にデータモデルの上部または離れて、ストリングまたはアレイで格納された（図８）。データモデルの作成時に、この情報は直接挿入されると判断され、データモデルにリファレンスポインタを有するアレイとしてそれを格納することができる。

【００９８】ライン１５～３９は、制御オブジェクト情報構築に関する第２のパスの間に、コードの１セクションが書き込まれた。ライン１５～２０は、トップレベルのオブジェクトの作成において用いられるコードを表す。トップレベルのオブジェクトは、図３に示すようなページオブジェクト３１４のようなページ全体のコンテナタイプのオブジェクトである。この制御オブジェクトは、作成されたASP+ページはこのタイプの制御オブジェクトを有するので、図１０に示す実質的なコードから比較的独立して構築される。

【００９９】図１１のライン２２～２７は、MyListと名

づけられた子制御オブジェクトの作成に用いられるコードを表す。図１０のライン１６に示すように、テーブルリスト制御が定義され、識別子"MyList"を与えられた。図１０のライン１６からの情報は、ステップ８１０(図８)において図１１に示されるライン２２～２７に翻訳される。本質的に、処理８１０において、図１０のライン１６のコード部分の構文解析すると、"tablelist"タイプの制御を作成すべきであると認識される。認識されると、図１１に示されるライン２２～２７の情報は、テーブルから検索するか、またはライン２２～２７に示されたコードを生成するための"MyList"のような適切な変数を挿入して公知の計算を用いて生成される。システムは、このタイプの制御を認識するように設計されているので、図１１に示されたコードを生成することができる。一旦、生成されると、処理８１０は、そのコードもファイルに書き込む。

【０１００】同様に、図１１のライン２９～３４は、別の制御オブジェクト、この場合は、"template"タイプのオブジェクトを表す。テンプレートは、それらもまたコンテナ制御であり、実行時間に実際に生成されるという点で特別なオブジェクトである。いずれにしても、テンプレートの制御オブジェクトの生成に必要なコードがステップ８１０（モデルにまだデータ構造が存在するかどうかを判断するテスト８１２に引き続いて）において生成される。上記のように、テンプレート制御オブジェクトが生成されるべきであり、図１１に示すコードが生成されファイルに書き込まれているという認識がなされる。重要なことは、図１１のライン３２および３３の"control3"へのリファレンスによって明らかなように、テンプレートのコードは、子制御に関連する情報を必要とする。したがって、データモデルが６０６(図６)で作成されると、この情報またはこの情報を決定する指標が、テンプレート制御情報とともに格納された。また、コードは、コードの生成が直接的であるようにテンプレート制御オブジェクトに対して予め決められている。

【０１０１】子制御が、図１０のライン１９のコードおよび図１１のライン３６～３９でのそれに対応する翻訳されたコードを用いてテンプレート内に作成される。図１０のライン１９は、識別子"MyLabel"を有する"label"タイプの制御を呼び出す。テーブルリスト制御およびテンプレート制御と同様に、ラベル制御は、処理８１０が認識する公知のタイプの制御であり、次いで適切なコードを生成し、それを図１１のライン３６～３９に示すようなファイルへ書き込むことができる。

【０１０２】データモデルの最後のパス時にライン４１～５１が作成および生成され、資源コードファイルに書き込まれた。ライン４１～５１におけるコードのラインは、クライアントへのレスポンスの一部になるHTMLコードをレンダリングするために呼び出されるレンダリングメソッドを表している。レンダリングコードは、ステップ８１６でのこのようなコードを生成すべきであるという認識に基づき生成される。一旦認識されると、コードは単に検索テーブルから判断されるか、または必要に応じて生成される。

【０１０３】図１１に示すコードは、特定の動的ウェブページファイル、すなわち図１０に示すファイルからのサーバ側生成コードを表している。図１１に示すファイルが完成すると、ファイルは、図６に示す処理６１０に関して上に述べたようにコンパイルされてもよい。コンパイルが 動的ウェブページコンテンツファイルの制御オブジェクト制御を生成するのに用いることができるクラスとなる。クラスは、キャッシュ、または他のメモリに格納され、ページのためのオブジェクトを例示するために所望の回数用いることができる。結果クラスのコンパイルおよびパーシスタンスのいずれも、ディスクへのアクセスが必要でないように「インメモリ」で行われ得、これは、一般にメモリで行うより時間がかかる。

【０１０４】サーバ側ページのセットアップ、起動および実行に必要なすべての処理は、動的にコンパイルされたページクラス内にカプセル化される。その結果、ページセットアップのあいだにファイルの追加的な構成／構文解析は必要でない。さらに、もともとの.aspx"またはASP+ファイルは再び扱われることはなく、「実行時間ホスト環境」、例えば、ASPスクリプトエンジンは、ページ実行処理の間要求されない。

【０１０５】本明細書中の本発明の実施形態は、１つ以上のコンピュータシステムの論理ステップとして実行される。本発明の論理処理は、（１）１つ以上のコンピュータシステムで実行処理されるプロセッサ実行ステップシーケンスとして、（２）１つ以上のコンピュータシステム内の相互接続された機械モジュールとして実行される。実行は選択の問題であり、本発明を実行するコンピュータシステムの性能要件に依存する。したがって、本明細書に記載の本発明の実施形態を構成する論理処理は、処理、ステップ、オブジェクトまたはモジュールと様々に表現することができる。

【０１０６】上記の明細書、実施例およびデータは、本発明の実施形態の構造および使用の完全な説明を提供している。本発明は、本発明の精神および範囲から逸脱することなく多数の形態で実施することができるので、本発明は添付の請求項にある。

【０１０７】

【発明の効果】

【図面の簡単な説明】

【図１】　本発明のある実施形態におけるクライアントに表示するウェブページコンテンツを動的に生成するウェブサーバを示す。

【図２】　本発明のある実施形態におけるサーバ側制御オブジェクトを用いてクライアント側ユーザインタフェース要素の処理およびレンダリングのための処理のフロ

ーチャートを示す。

【図３】　本発明のある実施形態で用いるウェブサーバのモジュールの一例を示す。

【図４】　本発明のある実施形態における動的コンテンツファイル（例えば、ASP+ページ）の一例を示す。

【図５】　本発明のある実施形態を実行するのに有用なシステムの一例を示す。

【図６】　本発明のある実施形態におけるページオブジェクトの処理を表すプロセスフローチャートを示す。

【図７】　データモデルを作成するためのサーバ側アプリケーションの構文解析を表すプロセスフローチャートを示す。

【図８】　資源コードファイルを作成するためのデータモデルのトラバースを表すプロセスフローチャートを示す。

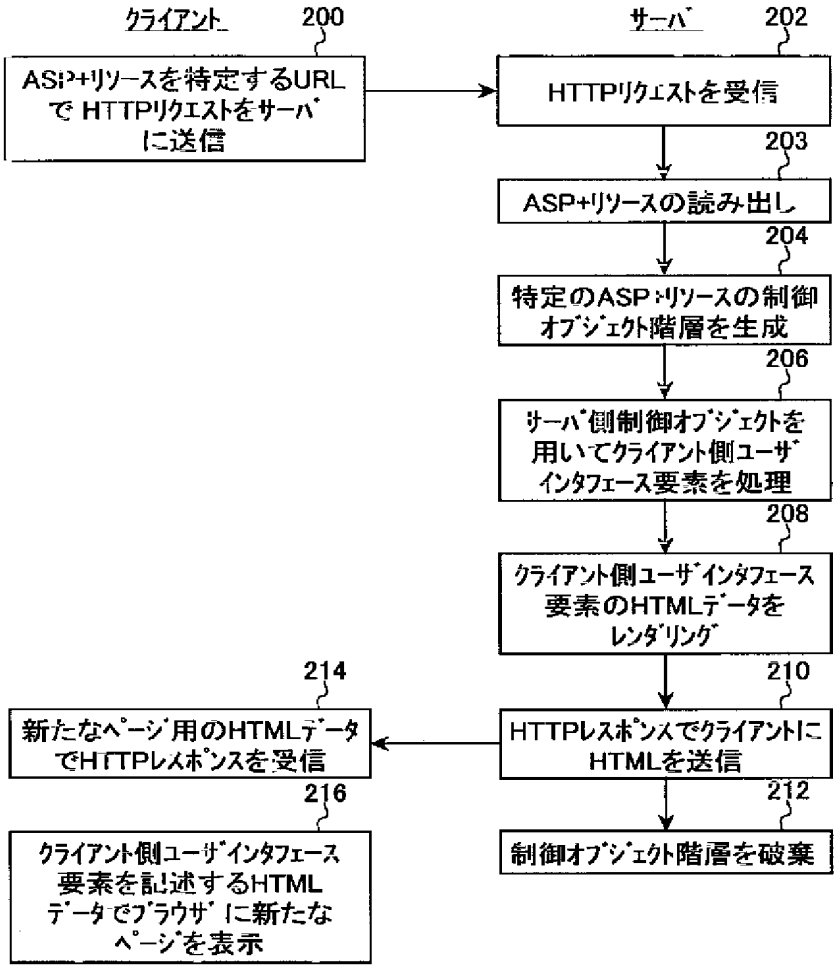
【図９】　本発明の別の実施形態における資源コードファイルを作成するためのデータモデルのトラバースを表すプロセスフローチャートを示す。

【図１０】　本発明による構文解析され得るASP+ページの一例を示す。

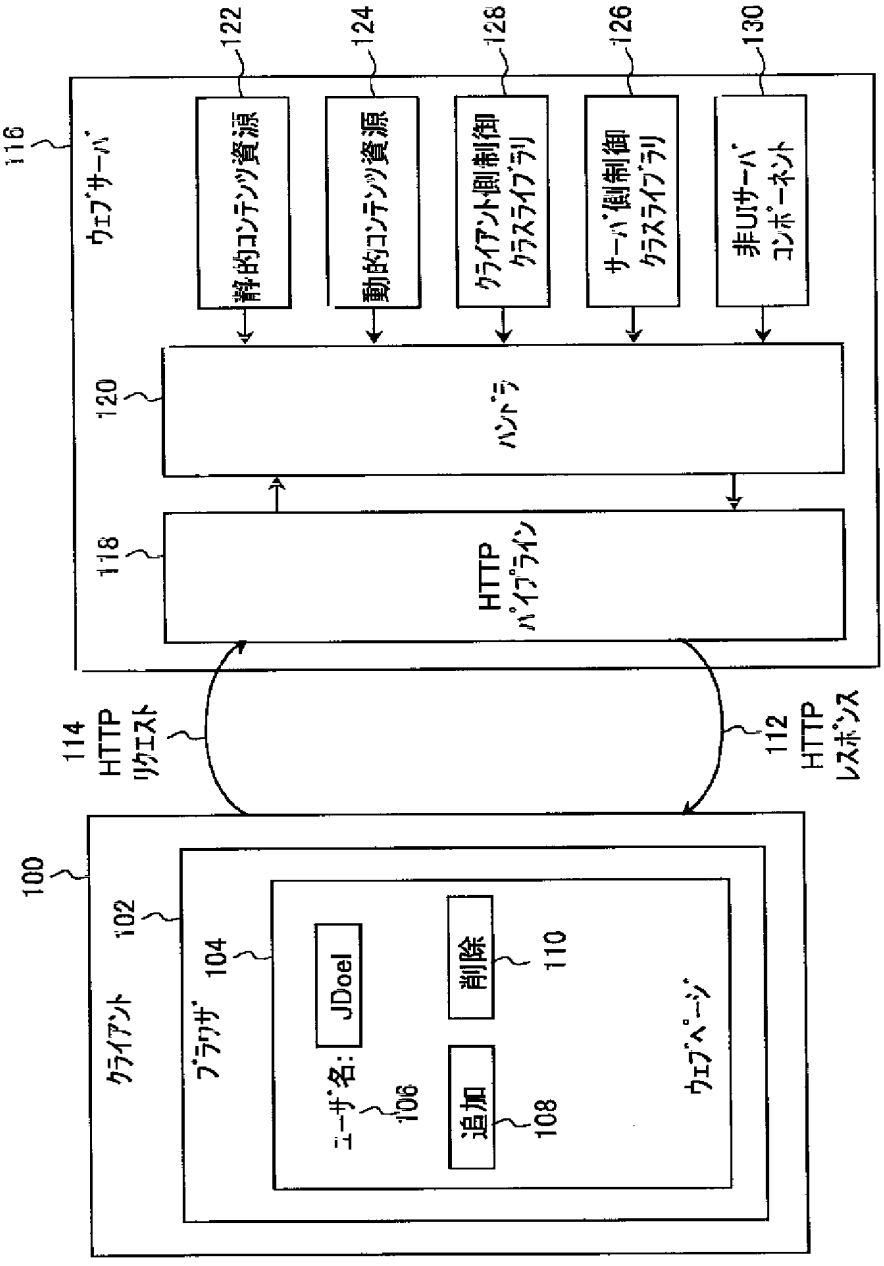
【図１１】　図１０に示す初期ASP+ファイルで本発明を用いて作成された資源コードファイルの一例を示す。

【符号の説明】

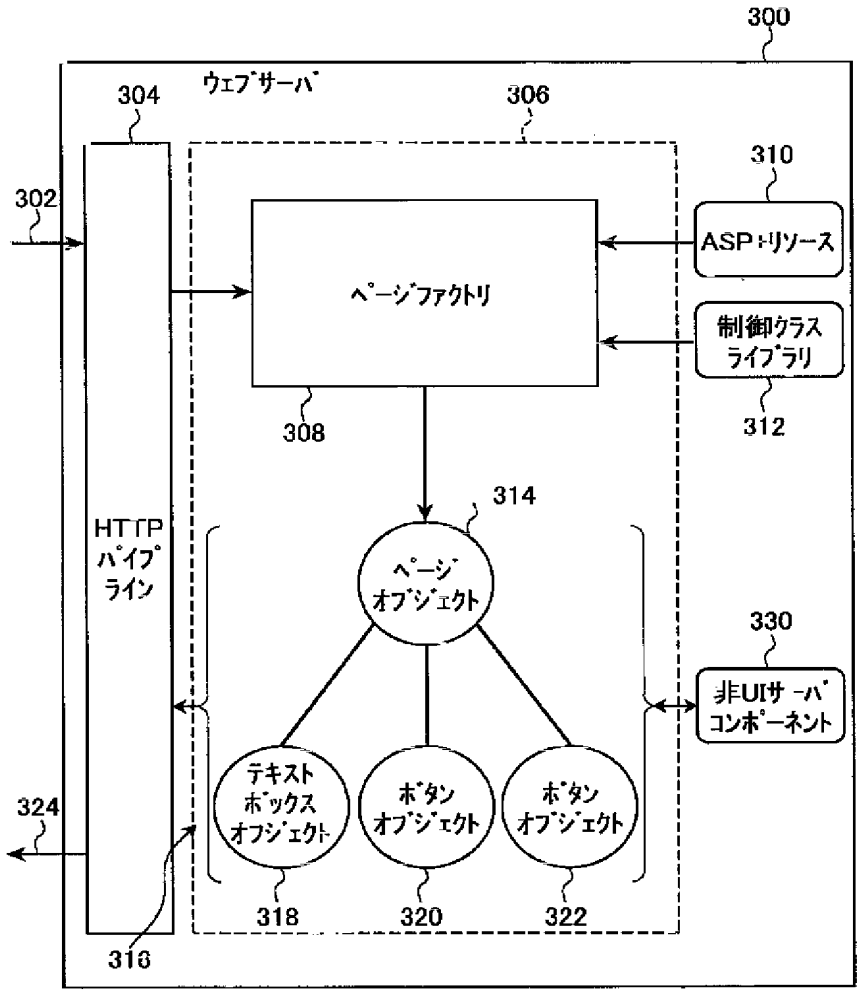
【図２】



【図1】


116
ウェブサーバ
122
静的コンテンツ資源
124
動的コンテンツ資源
128
クライアント側制御クラスライブラリ
126
サーバ側制御クラスライブラリ
130
非UIサーバコンポーネント
120
ハンドラ
118
HTTP パイプライン
114
HTTP リクエスト
112
HTTP レスポンス
100
クライアント
102
ブラウザ
104
ユーザ名:
JDoel
106
削除
110
追加
108
ウェブページ

【図３】



【図１０】

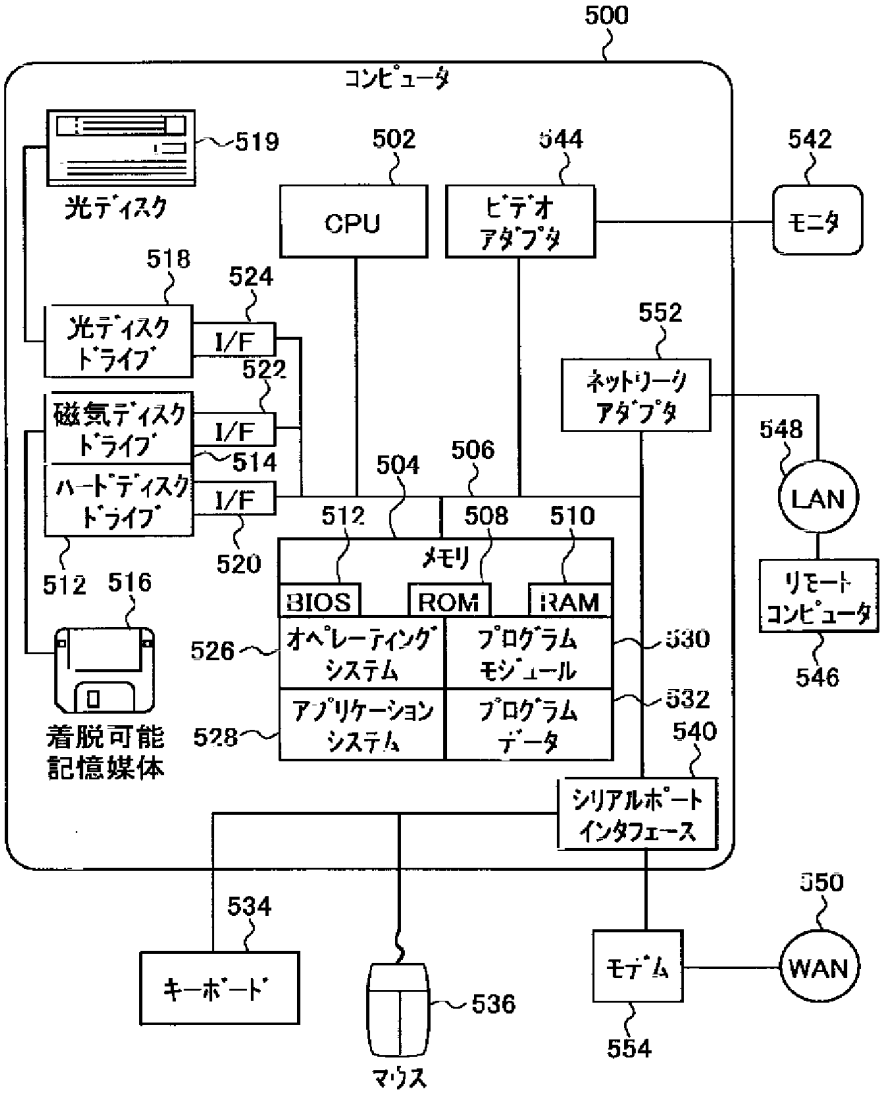
```
1     <%@ language="VB" Description="Test of the TableList control" %>
2
3     <script runat=server>
4        public MyData as New DataList
5
6        Overrides Sub Init()
7           MyData.Add "Name1"
8           MyData.Add "Name2"
9           MyData.Add "Name3"
10
11          Set MyList.DataSource = MyData
12       End Sub
13    </script>
14
15    <%%>
16    <wfc:TableList id="MyList" runat=server>
17       <template:ItemTemplate runat=server>
18    <%%>
19             <wfc:Label id="MyLabel" databinding="text:DataItem" runat=server/>
20       </template:ItemTemplate>
21    </wfc:TableList>
```

【図4】

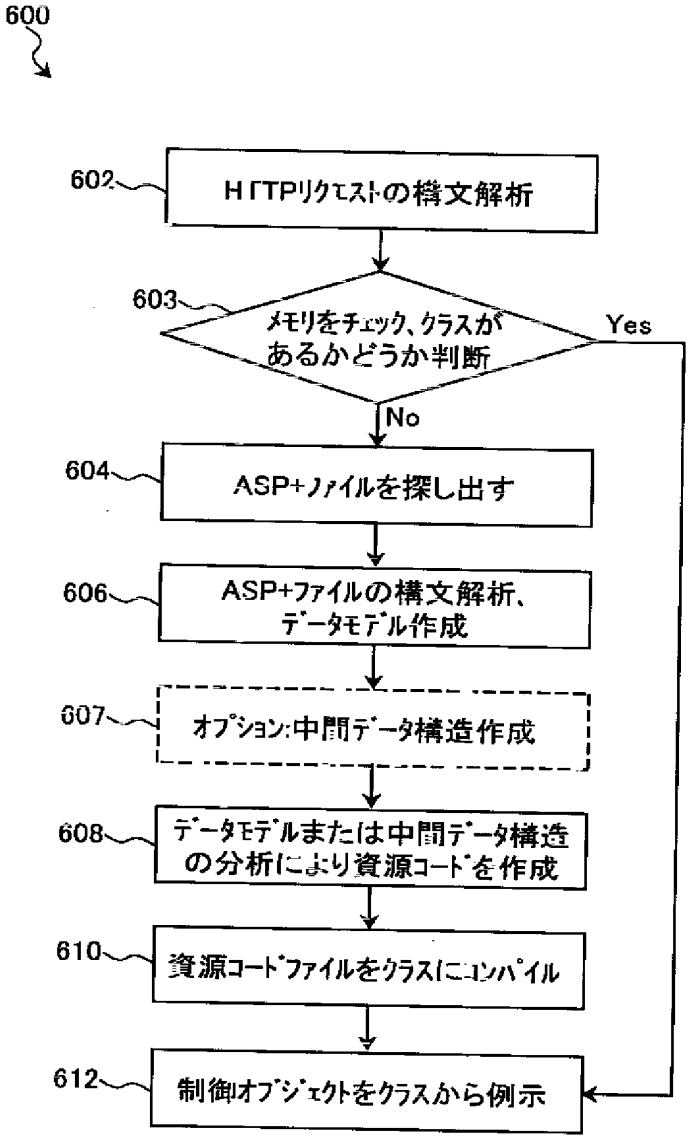
400
1 <%@ Page Language="VB" Description="Simple Sample Page" Errorpage="ErrorPage.aspx" %>
2 <html>
3 <script runat=server>
4 Sub AddButton_Click(ByVal Source as Object, By Val E as Event Args)
5 Message.Text = "Add" & UserName.Text
6 End Sub
7 Sub DeleteButton_Click(ByVal Source as Object, By Val E as Event Args)
8 Message.Text = "Delete" & UserName.Text
9 End Sub
10 </script>
11 <body>
12 <form runat="server">
13 User Name: <input type="Text" id="UserName" runat=server>
14
15 <button id="AddButton" value="ADD" OnClick="AddButton_Click" runat=server>
16 <button id="DeleteButton" value="DELETE" OnClick="DeleteButton_Click" runat=server>
17

18
19 </form>
20 </body>
21 </html>

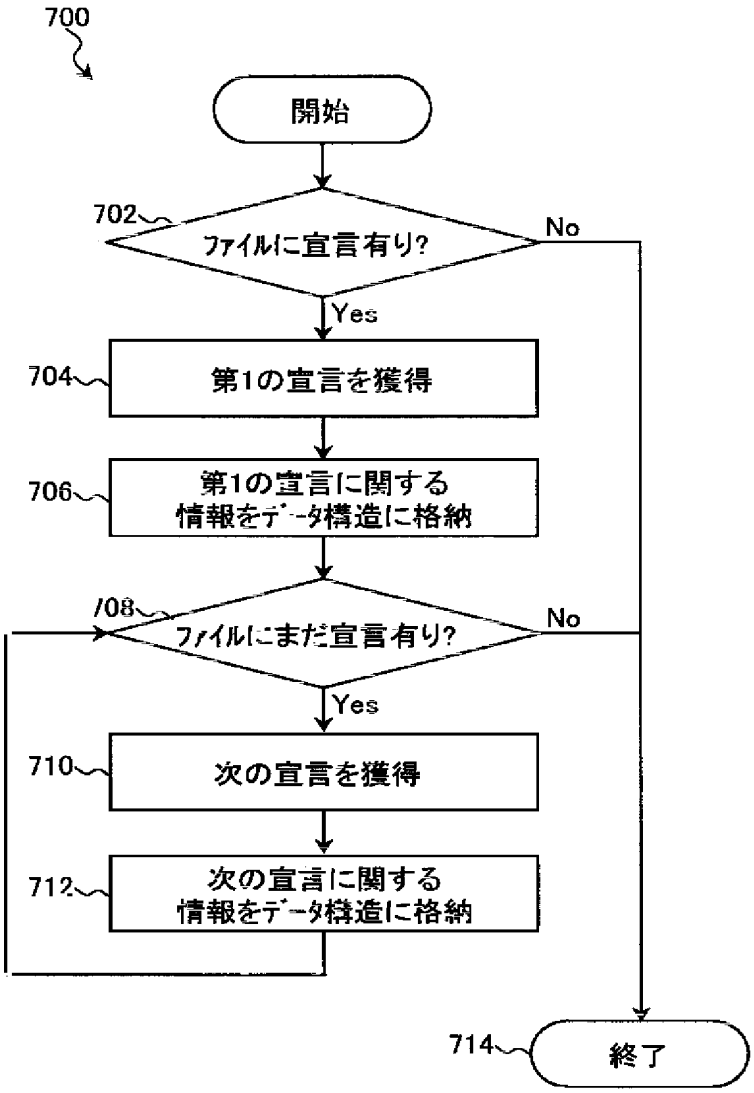
【図５】


500
コンピュータ
519
光ディスク
502
CPU
544
ビデオアダプタ
542
モニタ
518
光ディスクドライブ
524
I/F
522
磁気ディスクドライブ
I/F
514
ハードディスクドライブ
I/F
520
552
ネットワークアダプタ
548
LAN
リモートコンピュータ
546
504
506
512
508
510
メモリ
BIOS
ROM
RAM
512
516
着脱可能記憶媒体
526
オペレーティングシステム
プログラムモジュール
530
528
アプリケーションシステム
プログラムデータ
532
540
シリアルポートインタフェース
534
キーボード
536
マウス
モデム
550
WAN
554

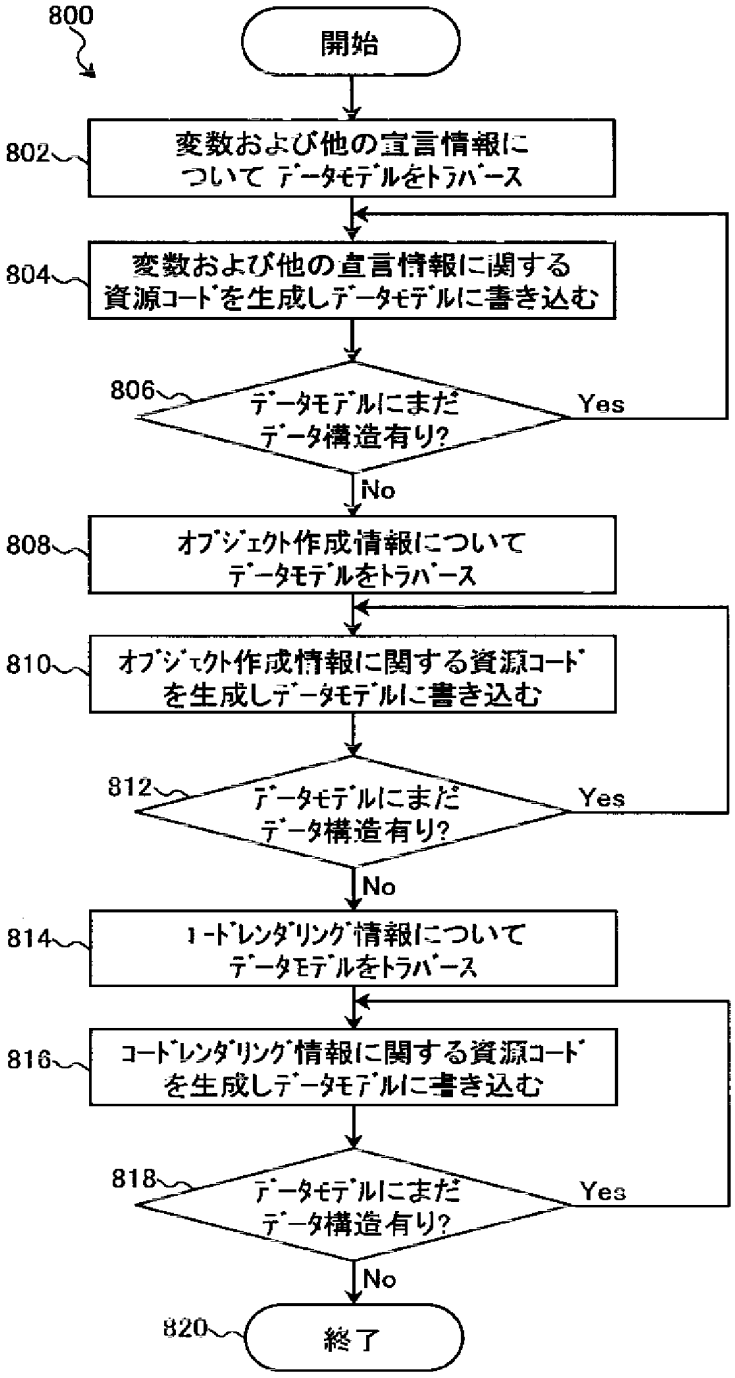
【図6】


600
602
HTTPリクエストの構文解析
603
メモリをチェック、クラスがあるかどうか判断
Yes
No
604
ASP+ファイルを探し出す
606
ASP+ファイルの構文解析、データモデル作成
607
オプション:中間データ構造作成
608
データモデルまたは中間データ構造の分析により資源コードを作成
610
資源コードファイルをクラスにコンパイル
612
制御オブジェクトをクラスから例示

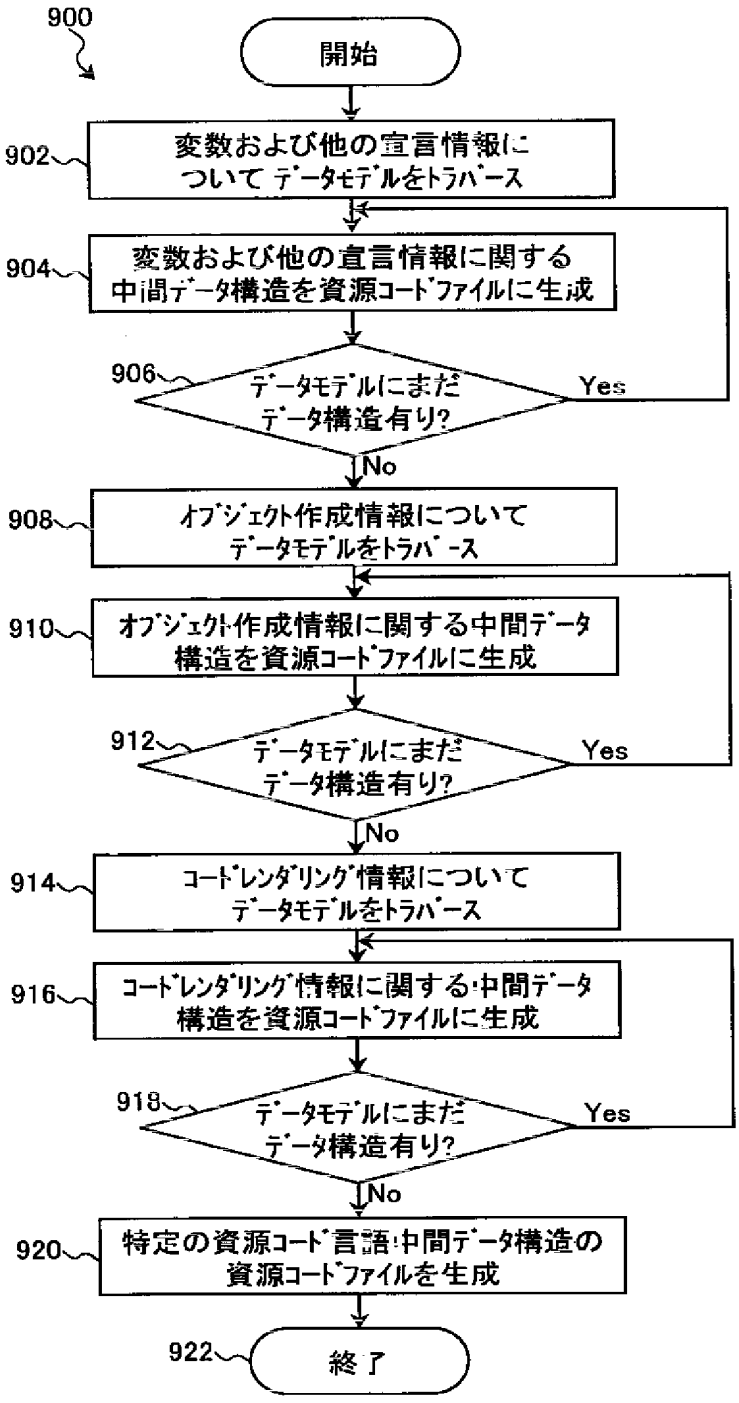
【図７】


700
開始
702
ファイルに宣言有り?
No
Yes
704
第1の宣言を獲得
706
第1の宣言に関する
情報をデータ構造に格納
708
ファイルにまだ宣言有り?
No
Yes
710
次の宣言を獲得
712
次の宣言に関する
情報をデータ構造に格納
714
終了

【図8】


800
開始
802
変数および他の宣言情報に
ついて データモデルをトラバース
804
変数および他の宣言情報に関する
資源コードを生成しデータモデルに書き込む
806
データモデルにまだ
データ構造有り?
Yes
No
808
オブジェクト作成情報について
データモデルをトラバース
810
オブジェクト作成情報に関する資源コード
を生成しデータモデルに書き込む
812
データモデルにまだ
データ構造有り?
Yes
No
814
リードレンダリング情報について
データモデルをトラバース
816
コードレンダリング情報に関する資源コード
を生成しデータモデルに書き込む
818
データモデルにまだ
データ構造有り?
Yes
No
820
終了

【図9】


900
開始
902
変数および他の宣言情報について データモデルをトラバース
904
変数および他の宣言情報に関する中間データ構造を資源コードファイルに生成
906
データモデルにまだデータ構造有り?
Yes
No
908
オブジェクト作成情報についてデータモデルをトラバース
910
オブジェクト作成情報に関する中間データ構造を資源コードファイルに生成
912
データモデルにまだデータ構造有り?
Yes
No
914
コードレンダリング情報についてデータモデルをトラバース
916
コードレンダリング情報に関する中間データ構造を資源コードファイルに生成
918
データモデルにまだデータ構造有り?
Yes
No
920
特定の資源コード言語中間データ構造の資源コードファイルを生成
922
終了

【図１１】

```
1   Inherits System.ASP.WebForms.Page
2
3   Dim MyList as TableList
4   Dim __control3 as Label
5
6   public MyData as New DataList
7
8   Overrides Sub Init()
9      MyData.Add "Name1"
10     MyData.Add "Name2"
11     MyData.Add "Name3"
12     Set MyList.DataSource = MyData
13  End Sub
14
15  Public Sub _tmp_aspx0__BuildControl__control0(ByVal __ctrl as ControlControl)
16     __ctrl.SetRenderMethodDelegate New RenderMethod(AddressOf
17  Me._tmp_aspx0 _Render__control0)
18     _tmp_aspx0__BuildControlMyList
19     __ctrl.AddParsedSubControl MyList
20  End Sub
21
22  Public Sub _tmp_aspx0__BuildControlMyList
23     set MyList = new TableList
24     MyList.ID = "MyList"
25     set MyList.ItemTemplate = new CompiledTemplateBuilder(new
26  BuildTemplateMethod(AddressOf me._tmp_aspx0__BuildControl _control2))
27  End Sub
28
29  Public Sub _tmp_aspx0__BuildControl__control2(ByVal __ctrl as ControlControl)
30     __ctrl.SetRenderMethodDelegate New RenderMethod(AddressOf
31  Me._tmp_aspx0__Render__control2)
32     _tmp_aspx0 _BuildControl__control3
33     __ctrl.AddParsedSubControl __control3
34  End Sub
35
36  Public Sub _tmp_aspx0__BuildControl__control3
37     set __control3 = new Label
38     __control3.ID = "MyLabel"
39  End Sub
40
41  Public Sub _tmp_aspx0__Render__control0(ByVal output as HtmlTextWriter,
42  ByVal __container as ControlControl)
43     Call __container.Controls(0).RenderControl(output)
44     output.Write(""&vbCRLF &""&vbCRLF &"")
45  End Sub
46
47  Public Sub _tmp_aspx0 _Render__control2(ByVal output as HtmlTextWriter,
48  ByVal __container as ControlControl)
49     Call __container.Controls(0).RenderControl(output)
50     output.Write(""&vbCRLF &"    ")
51  End Sub
```

【手続補正書】
【提出日】平成１３年６月１２日（２００１．６．１２）
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】０１０７
【補正方法】変更
【補正内容】
【０１０７】
【発明の効果】以上のように本発明によれば、中間言語または資源コードファイルをサーバ側資源から作成することができ、資源コードファイルは、実行処理可能なクラスにコンパイルされる。実行処理可能なクラスにより、クライアントレスポンスのレンダリングを含むサーバ側機能を実行するウェブページ制御オブジェクトの素早い生成が可能になる。
【手続補正２】
【補正対象書類名】明細書
【補正対象項目名】符号の説明
【補正方法】変更
【補正内容】
【符号の説明】
１００　クライアント
１０２　ブラウザ
１０４　ウェブページ

１０６　ユーザ名
１０８　追加
１１０　削除
１１２　HTTPレスポンス
１１４　HTTPリクエスト
１１６、３００　ウェブサーバ
１１８、３０４　HTTPパイプライン
１２０　ハンドラ
１２２　静的コンテンツ資源
１２４　動的コンテンツ資源
１２６　サーバ側制御クラスライブラリ
１２８　クライアント側制御クラスライブラリ
１３０、３３０　非UIサーバコンポーネント
３０８　ページファクトリ
３１０　ASP+リソース
３１２　制御クラスライブラリ
３１４　ページオブジェクト
３１８　テキストボックスオブジェクト
３２０、３２２　ボタンオブジェクト
５００　コンピュータ
５０２　ＣＰＵ
５０４　メモリ
５０８　ＲＯＭ
５１０　ＲＡＭ
５１２　ハードディスクドライブ
５１４　磁気ディスクドライブ
５１６　着脱可能記憶媒体
５１８　光ディスクドライブ
５１９　光ディスク
５２０、５２２、５２４　Ｉ／Ｆ
５２６　オペレーティングシステム
５２８　アプリケーションプログラム
５３０　プログラムモジュール
５３２　プログラムデータ
５３４　キーボード
５３６　マウス
５４０　シリアルポートインタフェース
５４２　モニタ
５４４　ビデオアダプタ
５４６　リモートコンピュータ
５４８　ＬＡＮ
５５０　ＷＡＮ
５５２　ネットワークアダプタ
５５４　モデム

フロントページの続き

(72)発明者　クーパー、ケネス　ビー.
アメリカ合衆国、98199　ワシントン州　シアトル、ウエスト　ブレイン　3410
(72)発明者　ギャスリー、スコット　ディー.
アメリカ合衆国、98052　ワシントン州　レッドモンド、アパートメント　ユー　2102、エヌイー　90ス　ストリート　17786
(72)発明者　エボ、デイビッド　エス.
アメリカ合衆国、98052　ワシントン州　レッドモンド、アパートメント　ジェイ　139、アボンデイル　ロード　10909
(72)発明者　アンダース、マーク　ティー.
アメリカ合衆国、98007　ワシントン州　ベルビュー、エヌイー　12ス　プレイス　14425
(72)発明者　ピータース、テッド　エー.
アメリカ合衆国、98119　ワシントン州　シアトル、８ス　アベニュー　ウエスト　2431
Ｆターム(参考)　5B076 DD04 DF08
5B081 AA09 CC41
5B082 HA02 HA08

【外国語明細書】

1 Title of Invention

SERVER-SIDE CODE GENERATION FROM A DYNAMIC WEB PAGE CONTENT FILE

2 Claims

1. In a server computer system having memory, a method of creating a class in memory, wherein the class is used by the server computer system to create server-side objects for dynamically rendering web page content, the web page content delivered to a client-side computer system and displayed as a web page on the client computer system, said method comprising:

receiving a request from the client specifying a dynamic web page content file;

processing the dynamic web page content file to produce a source code file containing source code that represent control objects declared in the web page content file; and

compiling the source code file to produce a class from which a set of hierarchical objects can be instantiated to produce web page authoring language that produces a web page for display.

2. A method as defined in claim 1 wherein the dynamic web page content file is a server-side declaration datastore.

3. A method as defined in claim 1 wherein the class is stored in cache memory on the server computer system and is available to instantiate objects in response to another request specifying the dynamic web page content file.

4. A method as defined in claim 1 wherein the class is stored on a magnetic storage medium and is available to instantiate objects in response to another request specifying the dynamic web page content file.

5. A method as defined in claim 1 wherein the step processing the dynamic web page content file comprises:

parsing the dynamic web page content file to store portions of the file into a data model, wherein the data model comprises a plurality of data objects linked in a hierarchical manner;

generating source code related to declaration information based on an

analysis of the data model during a first phase;

writing the source code related to declaration information to the source code file;

generating source code related to control object information based on an anlaysis of the data model during a second phase; and

writing the source code related to control object information to the source code file during the second phase.

6. A method as defined in claim 5 wherein the method further comprises:

generating source code related to rendering information based on an analysis of the data model during a third phase; and

writing the source code related to rendering information to the source code file during the third phase.

7. A method as defined in claim 6 wherein the second phase occurs once the first phase is substantially complete and wherein the third phase occurs once the second phase is substantially complete.

8. A method as defined in claim 6 wherein the first, second and third phases occurs substantially simultaneously.

9. A method as defined in claim 1 further comprising the following:

prior to the step of processing the dynamic web page content file, determining whether the class related to the received request has been compiled and stored in memory;

if the class has been compiled and stored in memory, skipping the processing step, otherwise continue with the processing step.

10. A computer data signal embodied in a carrier wave by a computing system having memory and encoding a computer program for executing a computer process creating a class in memory, wherein the class is used by the server computer system to create server-side objects for dynamically rendering web page content, the

web page content delivered to a client-side computer system and displayed as a web page on the client computer system, said computer process comprising:

receiving a request from the client specifying a dynamic web page content file;

processing the dynamic web page content file to produce a source code file containing source code that represent control objects declared in the web page content file; and

compiling the source code file to produce a class from which a set of hierarchical objects can be instantiated to produce web page authoring language that produces a web page for display.

11. A computer program storage medium readable by a computer system having memory and encoding a computer program for executing a computer process creating a class in memory, wherein the class is used by the server computer system to create server-side objects for dynamically rendering web page content, the web page content delivered to a client-side computer system and displayed as a web page on the client computer system, said computer process comprising:

receiving a request from the client specifying a dynamic web page content file;

processing the dynamic web page content file to produce a source code file containing source code that represent control objects declared in the web page content file; and

compiling the source code file to produce a class from which a set of hierarchical objects can be instantiated to produce web page authoring language that produces a web page for display.

12. In a server computer system having memory, a method of creating a plurality of web page responses having dynamically rendered web page content, the web page responses delivered to one or more client-side computer systems and displayed as a web pages on the client computer systems, said method comprising:

receiving a request from the client computer system for the web page, wherein the request identifies a dynamic web page content file;

creating a data model to store elements of the dynamic web page content file;

generating a source code file related to the dynamic web page content file based on the evaluation of the data model;

compiling the source code file to create a compiled class in memory;

returning a class reference to the server computer system enabling the server computer system to instantiate server-side processing objects from that class to dynamically generate web page content;

rendering the dynamic web page content into a web page response for delivery to the client computer system;

conducting the web page response to the requesting client computer system;

receiving a second request for the web page for the web page, wherein the request identifies a dynamic web page content file;

determining that a compiled class for that dynamic web page content file resides in memory;

returning a class reference to the server computer system enabling the server computer system to instantiate server-side processing objects from that class to dynamically generate web page content;

rendering the dynamic web page content into a second web page response; and

conducting the second web page response to the requesting client computer system.

13. A computer program storage medium readable by a computer system having memory and encoding a computer program for executing a computer process creating a plurality of web page responses having dynamically rendered web page content, the web page responses delivered to one or more client-side computer systems and displayed as a web pages on the client computer systems, said computer process comprising:

receiving a request from the client computer system for the web page, wherein the request identifies a dynamic web page content file;

creating a data model to store elements of the dynamic web page content file;

generating a source code file related to the dynamic web page content file based on the evaluation of the data model;

compiling the source code file to create a compiled class in memory;

returning a class reference to the server computer system enabling the server computer system to instantiate server-side processing objects from that class to dynamically generate web page content;

rendering the dynamic web page content into a web page response for delivery to the client computer system;

conducting the web page response to the requesting client computer system;

receiving a second request for the web page for the web page, wherein the request identifies a dynamic web page content file;

determining that a compiled class for that dynamic web page content file resides in memory;

returning a class reference to the server computer system enabling the server computer system to instantiate server-side processing objects from that class to dynamically generate web page content;

rendering the dynamic web page content into a second web page response; and

conducting the second web page response to the requesting client computer system.

14. A computer data signal embodied in a carrier wave by a computing system having memory and encoding a computer program for executing a computer process creating a plurality of web page responses having dynamically rendered web page content, the web page responses delivered to one or more client-side computer systems and displayed as a web pages on the client computer systems, said computer process comprising:

receiving a request from the client computer system for the web page, wherein the request identifies a dynamic web page content file;

creating a data model to store elements of the dynamic web page content file;

generating a source code file related to the dynamic web page content file based on the evaluation of the data model;

compiling the source code file to create a compiled class in memory;

returning a class reference to the server computer system enabling the server computer system to instantiate server-side processing objects from that class to dynamically generate web page content;

rendering the dynamic web page content into a web page response for delivery to the client computer system;

conducting the web page response to the requesting client computer system;

receiving a second request for the web page for the web page, wherein the request identifies a dynamic web page content file;

determining that a compiled class for that dynamic web page content file resides in memory;

returning a class reference to the server computer system enabling the server computer system to instantiate server-side processing objects from that class to dynamically generate web page content;

rendering the dynamic web page content into a second web page response; and

conducting the second web page response to the requesting client computer system.

15. A computer program product encoding a computer program for executing in a computer system a computer process for creating a class in memory, wherein the class is used by a server computer system to create server-side objects for dynamically rendering authoring language elements, the elements are delivered to a client-side computer system and processed on the client computer system, said process comprising:

receiving a request from the client computer system for the resource, wherein the request identifies a dynamic web page resource;

processing the resource to generate a source code file related to the resource;

compiling the source code file to create a compiled class in memory to enable the instantiation of objects of the compiled class.

16. A computer program product encoding a computer program for executing in a computer system a computer process for creating a class in memory as defined in claim 15, wherein the processing step of creating a data model comprises:

parsing the resource to separate the resource into logical elements and identify relationships between the logical elements;

creating a plurality of hierarchically related data structures forming a hierarchical data model;

storing portions of the resource in the data structures.

17. A computer program product encoding a computer program for executing in a computer system a computer process for creating a class in memory as defined in claim 15, wherein the processing step comprises the following steps:

performing a first analysis of the resource to generate source code related to variable declaration information;

performing a second analysis of the resource to generate source code related to control object information;

performing a third analysis of the resource to generate source code related to rendering information; and

storing the source code in the source code file.

18. A computer program product encoding a computer program for executing in a computer system a computer process for creating a class in memory as defined in claim 16, wherein the processing step of generating source code comprises further comprises the step of generating an intermediate data structure, wherein the source code is generated from the intermediate data structure.

19. A computer program product encoding a computer program for executing in a computer system a computer process for creating a class in memory as defined in claim 18, wherein the processing step of generating an intermediate data structure further comprises:

performing a first analysis of the resource to generate intermediate data structure elements related to variable declaration information;

performing a second analysis of the resource to generate intermediate data structure elements related to control object information;

performing a third analysis of the resource to generate intermediate data structure elements related to rendering information; and

generating source code from the intermediate data structure.

20. A computer program product encoding a computer program for executing in a computer system a computer process for creating a class in memory as defined in claim 20, wherein the intermediate data structure is a generic description that may be translated into a plurality of source code language files, wherein at least one source code file is different from another source code language file.

3 Detailed Description of Invention

Technical Field

The invention relates generally to a web server framework, and more particularly to server-side code generation to create control objects that process client-side user interface elements of a web page.

Background of the Invention

A typical web browser receives data from a web server defining the appearance and rudimentary behavior of a web page for display on a client system. In a typical scenario, a user specifies a Uniform Resource Locator ("URL"), a global address of a resource on the World Wide Web, to access a desired web site. An example URL is "http://www.microsoft.com/ms.htm". The first part of the example URL indicates a given protocol (i.e., "http") to be used in the communication. The second part specifies the domain name (i.e., "www.microsoft.com") where the resource is located. The third part specifies the resource (i.e., a file called "ms.htm") within the domain. Accordingly, a browser generates an HTTP (HyperText Transport Protocol) request associated with the example URL to retrieve the data associated with ms.htm file within the www.microsoft.com domain. A web server hosting the www.microsoft.com site receives the HTTP request and returns the requested web page or resource in an HTTP response to the client system for display in the browser.

The "ms.htm" file of the example above corresponds to a web page file that includes static HTML (HyperText Markup Language) code. HTML is a plain-text authoring language used to create documents (e.g., web pages) on the World Wide Web. As such, an HTML file can be retrieved from a web server by a client browser which converts the HTML code to actual visual images or audio components and is thus displayed as a web page. On the client computer systems, this process displays the web page content defined by the delivered HTML file. Using HTML, a developer can, for example, specify formatted text, lists, forms, tables, hypertext links, inline images and sounds, and background graphics for display in the browser. An HTML file, however, is a static file that does not inherently support dynamic

generation of web page content. Web page content is the HTML code that is returned to the client for display. Dynamic operation of such relates to a server side application that, as a result of processing steps, generates the HTML code prior to sending as opposed to just sending predetermined code to client browser.

In order to handle more complex client-server interaction, server-side application programs have developed to handle more complex client-server interaction, such as the providing of dynamic content, e.g., changing stock prices or traffic information. The server-side application program processes the HTTP request and generates the appropriate HTML code for transmission to the client in an HTTP response. For example, a server-side application program can process query strings and data from Web-based forms provided by the client in HTTP requests to dynamically generate HTML code for transmission in an HTTP response to the client. In essence, the server side application can generate an HTML-type file that is customized based on information in a request from a client. In such a case, there is no static HTML file that is stored on the server; the HTML file is dynamically created at runtime. An exemplary server-side application program may generate HTML code using a sequence of one or more formatted text write operations to a memory structure. Thereafter, the resulting text is transmitted to a client system in an HTTP response, where it is displayed in the browser.

Developing a server-side application program can be a complex task requiring not only familiarity with normal HTML coding that is used to layout a Web page, but also with programming basics, including one or more programming languages (e.g., C++, Perl, Visual Basic, or Jscript). Unfortunately however, many Web page designers are frequently graphics designers and editors, who provide the human touch but often lack programming experience. Thus, there is a need to provide a simplified web page development framework to create web page files that allows those with less programming experience to develop web page interfaces between server side applications and their respective clients. It is desirable, therefore, to provide a development framework to allow a developer to dynamically create and process a web page with minimal programming.

One approach to minimize the programming requirements of dynamic web page generation has been the Active Server Page (ASP) framework, provided by Microsoft Corporation. The ASP framework allows developers to create "ASP" web

page files that typically include Visual Basic or Jscript code, as well as other HTML code. The ASP file contains declarations or tags that perform various functions as well as VB script or J script code. These declarations are generally easier to write than writing actual programming code.

During operation, the HTTP request specifies the ASP file as the desired resource and, thereafter, the ASP file is used to generate the resulting HTML code in the HTTP response to the client. Furthermore, an ASP file may reference pre-developed or third party client-side library components (e.g., client-side ACTIVEX controls) as well as data bases or other third party applications to ease a given application programming efforts.

The simplified ASP web page file must be converted at runtime to a script that can be interpreted by a script engine. The script engine typically performs the various declaration-type commands in the ASP file in a consecutive or synchronous manner to achieve the desired result. Compared to files that are compiled and stored as executable files, script files run by script engines are generally slower since the script engine must perform an interpretation function rather than simply running the file.

One particular problem with compiling a script file into an executable file is that there is or maybe a combination of various languages in the script file. For example, the script file may include components written in HTML and others written in Visual Basic. The script engine uses various operations to interpret these elements at runtime, but no compilers exist to translate the different language components to a single language, i.e., a single source code file. Moreover, in the current server-side application frameworks, the programming required to dynamically manage client-side user interface elements (e.g., text boxes, list boxes, buttons, hypertext links, images, sounds, etc.) within server-side applications can still require sophisticated programming skills. As these server-side processes become more complex, script engines will not be able to continuously keep up with the demands.

For these and other reasons the present invention has been made.

Summary of the Invention

The present invention relates to a code generation method and apparatus to create an intermediate language or source code file from a server-side resource, the source code file then being compiled into an executable class. The executable class allows for rapid generation of web page control objects that perform server-side functions, including the rendering of client responses. In an embodiment of the present invention, the code generation scheme is capable of creating control objects connected in a hierarchy to handle event processing and the setting of attributes to the specific objects. Furthermore, the code generation method is also capable of connecting objects that have been declared using a template.

In accordance with preferred aspects, the present invention relates to a method of creating a class in a server computer system memory. The class is used by the server computer system to create server-side objects for dynamically rendering web page content and the web page content is delivered to a client-side computer system and displayed as a web page on the client computer system. In operation, the server computer system receives a request from the client computer system for the web page and wherein the request identifies a dynamic web page content file. The server computer creates a data model to store elements of the dynamic web page content file, evaluates the data model and generates a source code file related to the dynamic web page content file based on the evaluation of the data model. Once the source code file is created, the source code file is compiled to create a compiled class in memory. The process generally ends with the return of a class reference to the server computer system which enables the server computer system to use the class.

In accordance with other preferred embodiments, the method stores the class in cache memory on the server computer system. Once stored in cache memory, multiple server side page objects can be instantiated from the singular compiled class and the original resource is not used again. Each time a request for the web page is received, the server computer system determines whether a compiled class for that dynamic web page content file resides in memory. If the requested class does not exist in memory, it is created. Once the class is located, the server computer system instantiates server-side processing objects from that class to

dynamically generate web page content. The web page content is then rendered and conducted to the client computer system.

In accordance with yet other embodiments of the present invention, the method step of evaluating the data model involves the recursive traversal of the data model during a plurality of passes. During each pass, source code is generated and written to the source code file based on the evaluation of the data model during that pass. The data model is constructed using data structures that linked in a hierarchical manner.

An embodiment of a computer program product in accordance with the present invention includes a computer program storage medium readable by a computer system and encoding a computer program for executing a computer process of creating a compiled class in memory on the server computer. The compiled class is used to instantiate server-side processing object to render a response corresponding to a requested web page to be displayed on a client computer system. An alternative embodiment of a computer program product in accordance with the present invention includes a computer data signal embodied in a carrier wave by a computing system and encoding a computer program for creating the compiled class on the server.

Detailed Description of the Invention

An embodiment of the present invention relates to a method of creating a compiled class in memory for a particular web page defined by a dynamic web page content resource or file. Creating the compiled class involves creating a source code file from the web page file. The source code file is then compiled into the class. Once a compiled class exists in memory, a page object can be instantiated to render the response which is sent back to a client for display. The page object generally involves server-side control objects for processing and generating client-side user interface elements which are displayed on the web page. Furthermore, a hierarchy of server-side control objects can be declared in the web page file wherein these objects ultimately cooperate to generate the resulting authoring language code, such as HTML, for display of the web page on the client.

FIG. 1 illustrates a web server for dynamically generating web page content for display on a client in an embodiment of the present invention. A client 100 executes a browser 102 that displays a web page 104 on a display device of the client 100. The client 100 includes a client computer system having a display device, such as a video monitor (not shown). An "INTERNET EXPLORER" browser, marketed by Microsoft Corporation, is an example of a browser 102 in an embodiment of the present invention. Other exemplary browsers include without limitation "NETSCAPE NAVIGATOR" and "MOSAIC". The exemplary web page 104 includes a text box control 106 and two button controls 108 and 110. The

browser 102 receives HTML code in the HTTP response 112 from a web server 116 and displays the web page as described by the HTML code. Although HTML is described with reference to one embodiment, other authoring languages, including without limitation SGML (Standard Generalized Markup Language) and XML (eXtensible Markup Language), are contemplated within the scope of the present invention.

The communications between the client 100 and the web server 116 are conducted using a sequence of HTTP requests 114 and HTTP responses 112. Although HTTP is described with reference to one embodiment, other transport protocols, including without limitation S-HTTP, are contemplated within the scope of the present invention. On the web server 116, an HTTP pipeline module 118 receives HTTP request 114, resolves the URL, and invokes an appropriate handler 120 for processing the request. In an embodiment of the present invention, a plurality of handlers 120 to handle different types of resources are provided on the web server 116.

For example, if the URL specifies a static content file 122, such as an HTML file, a handler 120 accesses the static content file 122 and passes the static content file 122 back through the HTTP pipeline 118 for communication to the client 100 in an HTTP response 112. Alternatively, in an embodiment of the present invention, if the URL specifies a dynamic content resource or file 124, such as an "ASP+" (Active Server Page+) page, a handler 120 accesses the dynamic content file 124, processes the contents of the dynamic content file 124, and generates the resulting HTML code for the web page 104. Generally, a dynamic content resource, such as file 124, is a server-side declaration datastore that can be used to dynamically generate the authoring language code that describes a web page to be displayed on a client. The HTML code for the web page is then passed through the HTTP pipeline 118 for communication to the client 100 in an HTTP response 112.

During its processing, a handler 120 can also access libraries of pre-developed or third-party code to simplify the development effort. One such library is a server-side class control library 126, from which the handler 120 can instantiate server-side control objects for processing user interface elements and generating the resultant HTML data for display of a web page. In an embodiment of the present invention, one or more server-side control objects map to one or more user interface

elements, visible or hidden, on the web page described in the dynamic content file 124.

A handler 120 also has access to one or more non-user-interface server components 130 that execute on the web server 116 or on another accessible web server. A non-user-interface server component 130, such as a stock price look-up application or database component, may be referenced in or associated with a dynamic content file 124 that is processed by a handler 120. The non-user-interface server component 130 may process events raised by the server-side control objects declared in the dynamic content file 124. As a result, the processing provided by the server-side control objects simplifies the programming of the non-user-interface server component 130 by encapsulating the processing and generation of the user interface elements of a web page, which allows the developer of the non-user-interface server component 130 to concentrate on the specific functionality of the application, rather than on user interface issues.

FIG. 2 illustrates a flow diagram of operations for processing and generating client-side user interface elements using server-side control objects in an embodiment of the present invention. In operation 200, the client transmits an HTTP request to the server. The HTTP request includes a URL that specifies a resource, such as an ASP+ page. In operation 202, the server receives the HTTP request and invokes the appropriate handler for processing the specified resource. The ASP+ page is read in operation 203. Operation 204 generates a server-side control object hierarchy based on the contents of the specified dynamic content file (e.g., the ASP+ page).

In operation 206, the server-side control objects of the control object hierarchy perform one or more of the following operations: Postback event handling, postback data handling, state management, and data binding. In operation 208, each server-side control object in the hierarchy is called to generate (or render) data, such as HTML code, for display of client-side user interface elements in the web page. Note that, although the term "render" may be used to describe the operation of displaying graphics on a user-interface, the term "render" is also used herein to describe the operation of generating authoring language data that can be interpreted by client-application, such as browser, for display and client-side functionality. A more detailed discussion of the processing operation 206 and the

rendering operation 208 is provided in association with FIG. 6. Calls to render methods in individual control objects are performed using a tree traversal sequence. That is, a call to the render method of a page object results in recursive traversal throughout appropriate server-side control objects in the hierarchy.

Alternatively, the actual creation of the individual server-side control objects may be deferred until the server-side control object is accessed (such as when handling postback input, loading a state, rendering HTML code from the control object, etc.) in operations 206 or 208. If a server-side control object is never accessed for a given request, deferred control object creation optimizes server processing by eliminating an unnecessary object creation operation.

Operation 210 transmits the HTML code to the client in an HTTP response. In operation 214, the client receives the HTML code associated with a new web page to be displayed. In operation 216, the client system displays the user interface elements of the new page in accordance with the HTML code received from the HTTP response. In operation 212, the server-side control object hierarchy is terminated. Server-side control objects in the hierarchy are created in response to an HTTP request referencing an associated ASP+ page, and destroyed subsequent to the rendering of authoring language data (e.g., HTML data). Alternatively, operation 212 may alternatively be performed after operation 208 and before operation 210.

FIG. 3 illustrates exemplary modules in a web server used in an embodiment of the present invention. The web server 300 receives an HTTP request 302 into the HTTP pipeline 304. The HTTP pipeline 304 may include various modules, such as modules for logging of web page statistics, user authentication, user authorization, and output caching of web pages. Each incoming HTTP request 302 received by the web server 300 is ultimately processed by a specific instance of an Interface Handler; e.g., IHTTPHandler class (shown as handler 306). The handler 306 resolves the URL request and invokes an appropriate handler factory (e.g., a page factory module 308).

In FIG. 3, a page factory module 308 associated with the ASP+ page 310 is invoked to handle the instantiation and configuration of objects from the ASP+ page 310. The ASP+ page 310 is identified or referenced by a unique URL and may be further identified by ".aspx" suffix, although other suffixes may be used. When a

request for a particular ".aspx" resource is first received by the page factory module 308, the page factory module 308 searches the file system for the appropriate resource or file (e.g., the .aspx page 310). The file may contain text (e.g., authoring language data) or another data format (e.g., byte-code data or encoded data) that may later be interpreted or accessed by the server to service the request. If the physical file exists, the page factory module 308 opens the file and reads the file into memory. Alternatively, if the requested file exists but has been previously loaded into memory, then the resource may not necessarily need to be loaded into memory, as discussed in more detail below. If the requested aspx file cannot be found, the page factory module 308 returns an appropriate "file not found" error message, e.g., by sending an HTTP "404" message back to the client.

Once the ASP+ page 310 is read into memory, the page factory module 308 processes the file content to build a data model of the page (e.g., lists of script blocks, directives, static text regions, hierarchical server-side control objects, server-side control properties, etc.). The data model is used to generate a source code file of a new object class, such as a COM+ (Component Object Model+) class, that extends the page base class, which is the code that defines the structure, properties, and functionality of a page object. In an embodiment of the invention, the source listing is then dynamically compiled into an intermediate language, and later Just-In-Time compiled into platform native instructions (e.g., X86, Alpha, etc.). An intermediate language may include general or custom-built language code, such as COM+ IL code, Java bytecodes, Modula 3 code, SmallTalk code, and Visual Basic code. In an alternative embodiment, the intermediate language operations may be omitted, so that the native instructions are generated directly from the source listing or the source file (e.g., the ASP+ page 310). A control class library 312 may be accessed by the page factory module 308 to obtain predefined server-side control classes used in the generation of the control object hierarchy.

The page factory module 308 creates a page object 314 from the compiled class, wherein the page object 314, which is a server-side control object that corresponds to the web page 104 shown in FIG. 1. The page object 314 and its children (i.e., a text box object 318, a button object 320, and another button object 322) comprise an exemplary control object hierarchy 316. Other exemplary control objects are also contemplated in accordance with the present invention, including

without limitation objects corresponding to the HTML controls in Table 1 (discussed below), as well as custom control objects. The page object 314 logically corresponds to the web page 104 of FIG. 1. The text box object 318 corresponds to the text box 106 in FIG. 1. Likewise, the button object 320 corresponds to the add button 108 in FIG. 1, and the button object 322 corresponds to the delete button 110 in FIG. 1. The page object 314 is hierarchically related to other control objects on the server. In one embodiment, a page object is a container object that hierarchically contains its children objects. In an alternative embodiment, other forms of hierarchical relations may be employed, including a dependency relationship. In a more complex control object hierarchy with multiple levels of children, a child object can be a container object for other child objects.

In the illustrated embodiment, the control objects in the control object hierarchy 316 are created and executed on the server 300, and each server-side control object logically corresponds to a corresponding user-interface element on the client. The server-side control objects also cooperate to handle postback input from the HTTP request 302, to manage the states of server-side control objects, to perform data binding with server-side databases, and to generate authoring language data (e.g., HTML code) used to display a resulting web page at the client. The resulting authoring language data is generated (i.e., rendered) from the server-side control object hierarchy 316 and transmitted to the client in an HTTP response 324. For example, resulting HTML (or other authoring language) code may reference ACTIVEX-type controls or other HTML constructs that yield client-side user interface elements (e.g., control buttons, text boxes, etc.) when processed by a browser.

By virtue of declarations made in the ASP+ page 310, server-side control objects may access one or more non-user-interface server components 330 to provide interaction between the non-user-interface server component 330 and client-side user interface elements. For example, in response to postback input, server-side control objects can raise server-side events to the non-user-interface server components registered for those events. In this manner the non-user-interface server component 330 can interact with the user through user interface elements without programming the code required to display and process these elements.

FIG. 4 illustrates contents of an exemplary dynamic content file in an embodiment of the present invention. In the illustrated embodiment, the file 400 contains plain-text declarations in an exemplary dynamic content file format (e.g., ASP+). Each declaration provides instructions to the page factory module 308 that reads the file 400, creates the class, invokes the appropriate server-side control objects which ultimately render HTML code or any other authoring language for transmission to the client in an HTTP response.

The first line of the file 400 includes a directive between delimiters *"<%@"* and *"%>"* in the format:

<%@ directive {attribute=value} %>

where *directive* may include without limitation "page", "cache" or "import". Directives are used by the page factory module 308 when processing a dynamic content file to determine such characteristics as buffering semantics, session state requirements, error handling schemes, scripting languages, transaction semantics, and import directions. Directives may be located anywhere within a page file.

In the second line, <html> is a standard HTML starting tag, which is written to the source code file as a literal text such that no additional processing takes place on the information in order to render the resulting HTML code other than a straightforward "write" command. In HTML, the <html> indicates the beginning of the HTML file and is paired with the closing tag on line 21, </html>, which is also a literal.

A code declaration block is located at lines 3-10 of the file 400. Generally, code declaration blocks define page objects and control object member variables and methods that are executed on the server. In the format:

<script runat = "server" [language = "*language*"][src = "*externalfile*"]>

.....

</script>

where the language and src parameters are optional. In an embodiment of the present invention, code declaration blocks are defined using <script> tags that contain a "runat" attribute having a value set to "server". Optionally, a "language" attribute may be used to specify the syntax of the inner code. The default language may represent the language configuration of the overall page; however, the

"language" attribute in the code declaration block allows a developer to use different languages within the same web page implementation including, for example, Jscript and PERL (Practical Extraction and Report Language). The "<script>" tag may also optionally specify a "src" file, which is an external file from which code is inserted into the dynamic content file for processing by the page compiler. It should be understood that the disclosed syntax is used in one embodiment, however, alternative embodiments may employ different syntaxes within the scope of the present invention.

In FIG. 4, two subroutines are declared in Visual Basic format within the code declaration block: AddButton_Click and DeleteButton_Click. Both subroutines take two input parameters, "Source" and "E", and are called when a client-side click event is detected on the corresponding button. In the AddButton_Click subroutine, the text in a UserName text box is concatenated on to the word "Add" and loaded into the Text data member of Message. In the DeleteButton_Click subroutine, the text in the UserName text box is concatenated on to the word "Delete" and loaded into the Text data member of Message. Although not shown in FIG. 4, member variables of server-side control objects may be declared in the code declaration block of the file 400. For example, using a Visual Basic syntax, the key word "DIM" declares a data variable of a server-side control object.

A "code render block" (not shown) can also be included in a dynamic content file. Code render blocks can contain an arbitrary amount of code that executes at page render time. In an embodiment of the present invention, a code render block executes in a "rendering" method that executes at page render time. Other portions of code can be executed in that rendering method. A code render block satisfies the following format (although other formats are contemplated in alternative embodiments):

<% InlineCode %>

where *InlineCode* includes self-contained code blocks or control flow blocks that execute on the server at page render time.

Inline expressions may also be used within an expression render block delimiters "*<%=*" and "*%>*" using the exemplary syntax:

<%= InlineExpression %>

where the expression contained in an *InlineExpression* block is ultimately encompassed by a call to "Response.Write(*InlineExpression*)" in a page object, which writes the value resulting from *InlineExpression* into appropriate place holder in the declaration. For example, *InlineExpressions* may be included in file 400 as follows:

<font size = "<% =x%>" > Hi <%=Name%>, you are <%=Age%>!

which outputs a greeting and a statement about a person's age in a font stored in the value "x". The person's name and age are defined as strings in a code declaration block (not shown). The resulting HTML code is rendered at the server for transmission to the client in the HTTP response so as to include the values of the *InlineExpressions* at appropriate locations:

<font size = "12"> Hi Bob, you are 35!

On line 11 of file 400, <body> is a standard HTML tag for defining the beginning of the body of the HTML document. On line 20 of file 400, the closing tag, </body>, is also shown. Both the <body> and </body> are literals in an embodiment of the present invention.

Within the body section of the HTML file 400, on line 12, the starting tag, <form>, of an HTML form block is found in FIG. 4. The ending tag, </form>, of the form block is found on line 19 of the HTML file 400. An optional parameter "id" may also be included in the HTML control tag, <form>, to associate a given identifier with the form block, thereby allowing multiple form blocks to be included in a single HTML file.

On line 18 of file 400, a server-side label, identified by "Message", is declared. The "Message" label is used in the code declared at lines 5 and 8 of the file 400 to display a label on the web page.

Within a form block, three exemplary HTML control tags are shown, corresponding to the user interface elements 106, 108, and 110 of FIG. 1. The first user interface element is declared on line 13 of the file 400 corresponding to a textbox. The text literal "User Name:" declares a label positioned to the left of the textbox. The input tag with type="Text" declares a textbox server-side control object having an id equaling "UserName" as a server-side control object that renders a textbox client-side user interface element. Lines 15 and 16 of file 400 declare the client-side user interface elements shown as buttons 108 and 110 of FIG. 1,

respectively. Note that the "OnClick" parameter specifies the appropriate sub-routine declared in the code declaration block of the file 400. As such, the server-side button control objects generated in response to the declarations in file 400 render the HTML code for the client-side buttons and an associated script for implementing the button click events.

The textbox and buttons declared in file 400 are examples of HTML server control declarations. By default, all HTML tags within an ASP+ page are treated as literal text content and are programmatically inaccessible to page developers. However, page developers can indicate that an HTML tag should be parsed and treated as an accessible server control declaration by marking it with a "runat" attribute with a value set to "server". Each server-side control object may optionally be associated with a unique "id" attribute to enable programmatic referencing of the corresponding control object. Property arguments and event bindings on server-side control objects can also be specified using declarative name/value attribute pairs on the tag element (e.g., the OnClick equals "MyButton_Click" pair).

In an embodiment of the present invention, a general syntax for declaring HTML control objects is as follows:

<HTMLTag id = "Optional Name" runat = server>

.....

</HTMLTag>

where the *OptionalName* is a unique identifier for the server-side control object. A list of HTML tags and the associated syntax and COM+ class that may be supported are illustrated in TABLE 1, although other HTML tags are contemplated within the scope of the present invention.

HTML_Tag Name	Example	COM+ Class
<a>	 My Link 	AnchorButton
<img>	<img id = "MyImage" runat = server>	Image
		Label
	Some contents	Panel
<form>	<form id = "MyForm" runat = server> </form>	FormControl
<select>	<select id = "MyList" runat = server> <option>One</option> <option>Two</option> <option>Three</option> </select>	DropDownList
<input type = file>	<input id = "MyFile" type = file runat = server>	FileInput
<input type = text>	<input id = "MyTextBox" type = text>	TextBox
<input type = password>	<input id = "MyPassword" type = password>	TextBox
<input type = reset>	<input id = "MyReset" type = reset>	Button
<input type = radio>	<input id = "MyRadioButton" type = radio runat = server>	RadioButton
<input type = checkbox>	<input id = "MyCheck" type = checkbox runat = server>	CheckBox
<input type = hidden>	<input id = "MyHidden" type = hidden runat = server>	HiddenField
<input type = image>	<input type = image src = "foo.jpg" runat = server>	ImageButton
<input type = submit>	<input type = submit runat = server>	Button
<input type = button>	<input type = button runat = server>	Button
<button>	<button id = MyButton runat = server>	Button
<textarea>	<textarea id = "MyText" runat = server> This is some sample text </textarea>	TextArea

TABLE 1

In addition to standard HTML control tags, an embodiment of the present invention enables developers to create re-usable components that encapsulate common programmatic functionality outside of the standard HTML tag set. These custom server-side control objects are specified using declarative tags within a page

file. Custom server-side control object declarations include a "runat" attribute with a value set to "server". Optionally, the unique "id" attribute may be specified to enable programmatic referencing of the custom control object. In addition, declarative name/value attribute pairs on a tag element specify property arguments and event bindings on a server-side control object. In-line template parameters may also be bound to a server-side control object by providing an appropriate "template" prefix child-element to the parent server control object. A format for a custom server-side control object declaration is:

<servercntrlclassname id="*OptionalName" [propertyname="propval"]* runat=server/>

where *servercntrlclassname* is a name of an accessible server control class, *OptionalName* is a unique identifier of the server-side control object, and *propval* represents an optional property value in the control object.

Using an alternative declaration syntax, XML tag prefixes may be used to provide a more concise notation for specifying server-side control objects within a page, using the following format:

<tagprefix:classname id = *"OptionalName"* runat = server/>

where *tagprefix* is associated with a given control name-space library and *classname* represents a name of a control in the associated name-space library. An optional *propertyvalue* is also supported.

With reference to FIG. 5, an exemplary computing system for embodiments of the invention includes a general purpose computing device in the form of a conventional computer system 500, including a processor unit 502, a system memory 504, and a system bus 506 that couples various system components including the system memory 504 to the processor unit 500. The system bus 506 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 508 and random access memory (RAM) 510. A basic input/output system 512 (BIOS), which contains basic routines that help transfer information between elements within the computer system 500, is stored in ROM 508.

The computer system 500 further includes a hard disk drive 512 for reading from and writing to a hard disk, a magnetic disk drive 514 for reading from or

writing to a removable magnetic disk 516, and an optical disk drive 518 for reading from or writing to a removable optical disk 519 such as a CD ROM, DVD, or other optical media. The hard disk drive 512, magnetic disk drive 514, and optical disk drive 518 are connected to the system bus 506 by a hard disk drive interface 520, a magnetic disk drive interface 522, and an optical drive interface 524, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, programs, and other data for the computer system 500.

Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 516, and a removable optical disk 519, other types of computer-readable media capable of storing data can be used in the exemplary system. Examples of these other types of computer-readable mediums that can be used in the exemplary operating environment include magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), and read only memories (ROMs).

A number of program modules may be stored on the hard disk, magnetic disk 516, optical disk 519, ROM 508 or RAM 510, including an operating system 526, one or more application programs 528, other program modules 530, and program data 532. A user may enter commands and information into the computer system 500 through input devices such as a keyboard 534 and mouse 536 or other pointing device. Examples of other input devices may include a microphone, joystick, game pad, satellite dish, and scanner. These and other input devices are often connected to the processing unit 502 through a serial port interface 540 that is coupled to the system bus 506. Nevertheless, these input devices also may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor 542 or other type of display device is also connected to the system bus 506 via an interface, such as a video adapter 544. In addition to the monitor 542, computer systems typically include other peripheral output devices (not shown), such as speakers and printers.

The computer system 500 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 546. The remote computer 546 may be a computer system, a server, a router, a network PC, a peer device or other common network node, and typically includes

many or all of the elements described above relative to the computer system 500. The network connections include a local area network (LAN) 548 and a wide area network (WAN) 550. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, the computer system 500 is connected to the local network 548 through a network interface or adapter 552. When used in a WAN networking environment, the computer system 500 typically includes a modem 554 or other means for establishing communications over the wide area network 550, such as the Internet. The modem 554, which may be internal or external, is connected to the system bus 506 via the serial port interface 540. In a networked environment, program modules depicted relative to the computer system 500, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary, and other means of establishing a communication link between the computers may be used.

In an embodiment of the present invention, the computer 500 represents a web server, wherein the CPU 502 executes a page factory module 308 on an ASP+ file stored on at least one of storage media 516, 512, 514, 518, 519, or memory 504. HTTP responses and requests are communicated over the LAN 548 which is coupled to a client computer 546.

The process 600 performed by the page factory module 308 is illustrated in the flow chart shown in FIG. 6 where process 600 generally corresponds to the process of Generating a Control Object Hierarchy 204 (FIG. 2). Process 600 converts an ASP+ page (also referred to as an ASP+ file) into a compiled, object-code class which is then loaded into memory to instantiate the control objects 314, 318, 320 and 322 (FIG. 3).

The page creation process 600 begins after the server receives a request for a URL at operation 202 (FIG. 2). Following the receipt of the request, parse operation 602 parses the requested URL to determine which resource is requested. Resources may include static files (.htm, .gif, .jpg, etc.), directory browsing activation, DAV (digital audio/video) files as well as dynamic content requests (.aspx, .soap, etc), which the present invention is designed to handle. Incoming HTTP requests received by the server are ultimately processed by a specific instance of a handler

class, e.g., an IHTTPHandler class. Through the use of the handler factories, e.g., IHTTPHandler "factories," a pluggable architecture is provided that resolves the actual resolution of URL requests to handler instances, e.g., IHTTPHandler instances. This resolution is facilitated using application configuration settings that can map an incoming URL's file-extension and an HTTP command to a factory class, e.g., an IHTTPHandlerFactory class ultimately responsible for creating an appropriate handler instance, e.g., an IHTTPHandler instance. Although the parse operation can identify a variety of resources through the pluggable architecture, the remainder of the discussion will focus on an HTTP request that identifies a dynamic web page content resource, and more particularly on a dynamic web page content file, such as an .aspx or ASP+ page or file.

Once the actual resource is identified at parse operation 602, check operation 603 checks the memory to determine if the class exists in memory. In order to determine whether the class has been compiled either check operation 603 searches for the class in memory by name by searching for a flag that was set when the class was compiled. Either way check operation 603 searches for an indicator that the compiled class has already been compiled and stored. If the class is in memory, flow branches YES to instantiate operation 612. Instantiate operation instantiates control objects from the compiled class. If check operation 603 determines that the class is not being compiled, then flow branches NO to locate operation 604.

If check operation 603 determines that the class does not exist in memory, locate operation 604 searches for and locates the specific resource and reads the file into memory. A base class, such as "System.ASP.WebForms.PageFactory" class, provides a handler factory implementation that handles the instantiation and configuration of ASP and ASPX pages. If the resource, in this case a physical dynamic web page content file, is not found, an appropriate error message is returned.

Following the locate operation 604, the process 600 continues with parse create operation 606, which parses the resource, such as an ASP+ file. The parse/create operation 606, reads the ASP+ file, declaration by declaration, and creates a data model from information gleaned while parsing. A data model is data structure, containing elements that are related to the elements of the active content file from which source code can be derived. The data model contains structural

elements that were referenced in the active content file and these elements are connected so as to represent the structure of the resulting control objects of the active server page. In an embodiment of the invention, the data model is a combination of objects that are linked in a hierarchical tree structure.

Each declaration of the web content file has certain elements that indicate how to create the data model as well as information to be stored in the data model. For example, if the declaration is a literal text declaration then the data model only needs to contain the information in such a manner that the text will be inserted in the proper location. However, if the declaration indicates that there is a nested control object, then the data model must be created that follows the nesting as declared in the ASP+ file. In essence, the data model will contain an object for each of the declarations, and each object will contain information related to whether child objects exist for each object.

Once the ASP+ file has been parsed and the data model has been created, an optional create operation 607 may be performed. Create operation 607 analyzes the data model and creates an intermediate data structure which is an abstraction of the source code, e.g., the source code file that is to be created, as discussed below. In essence the intermediate data structure is a generic data structure that describes code. The intermediate data structure has an upper level data structure that describes the classes. For each class there may be another level data structure that has the class name and a list or array of the methods. Further, each method is itself a data structure having various elements such as statements and declarations. The statement may then include items such as expressions, data calls, etc.

Once the ASP+ file has been parsed and the data model (and optionally, the intermediate data structure) has been created, create source code module 608 creates the necessary source code through an analysis of the data model, typically writing various lines of code for each declaration found in the ASP+ file. In general, this step involves the straightforward translation of known declarations and other information in the data model into code. Such translations are either hard coded into the application that performs the translation or each translation may be stored in a lookup table. Lookup tables of this sort can be created to supply the proper translation for almost any language, yet the use of object oriented source code languages simplifies the compilation into COM or COM+ classes to take advantage

of existing COM or COM+ libraries. (Exemplary source code languages include C++ or VBasic.) Moreover, if an intermediate data structure is created at operation 607, then the translation becomes even more straightforward. In such a case, the data structure is significantly closer to being a generic source code language file such that the translation may be merely the addition of proper lexicon and syntax to create a source code file for a particular source code language.

The ASP+ file declares a particular source code language for the resulting source code file. If no language has been declared in the ASP+ file, then a default language, such as Visual Basic, may be used. The resulting source code file, generated from the data model or from the intermediate data structure, is essentially the type of sophisticated source code file a programmer would have to write if that programmer was to take advantage of the server-side control object hierarchy and did not have the benefit of this code generation tool.

Upon completion of the source code file, compile operation 614 compiles the source code file, creating a compiled class in object code or other executable form. Alternatively, the class could be compiled into byte codes or other language that can run on a virtual machine. Compiling the source code file involves the use of a compiler designed to compile the source code language found in the source code file. The class may then be called by instantiate operation 616 to create the page object and resulting control objects for the web page. The instantiate operation, if not explicitly provided in the URL request is an implicit operation as part of the URL request for a particular resource.

Once the ASP+ file has been compiled into a page class, the class remains available for future requests. Therefore, the process of generating source code for the ASP+ file only needs to happen once and later requests can take advantage of the compiled class, instantiating necessary objects as required. The compiled class may be cached for a short period of time and/or the compiled class may be stored to disk. Indeed, once the ASP+ file has been compiled, it is not necessary to touch the original ASP+ file again. Of course if the ASP+ file is modified, then the compilation process is repeated to update the page class. An appropriate flag or other indicating mechanism can be used to determine whether the ASP+ file has been updated, therefore enabling automatic class updates. Alternatively, the

responsibility of removing the cached class upon updating the ASP+ up to the web page file can be left to the developer.

Details associated with the creation of the data model are shown in FIG. 7. The data model creation process 700 begins as test operation 702 determines whether there are any declarations in the ASP+ file. If there are no declarations to be compiled, such as in a pure HTML file, flow branches NO to the end of the process 714.

If there is at least one declaration in the ASP+ file, as determined at test operation 702, flow branches YES to get operation 704, which gets the first declaration in the ASP+ file. The information from the first declaration is then stored in a data structure by store operation 706. Not only is the actual declaration stored in the data structure, but also information related to the first declaration can be stored in the data structure such as whether the declaration declares a container type control having children or whether the declaration declares directives, etc. Since the web page is an ASP+ file, it typically has, as a first declaration, a directives tag. The directives tag includes various directives, e.g., information between the delimiters "<%@" and the "%>" as shown in line 1 in FIG. 4. It should be understood that the delimiters shown in FIG. 4 are merely exemplary and other choices are possible. The directives supply information to the page factory module that is used in creating the source code file. For example, the directives shown in FIG. 4 at line 1 indicate, for example, that the default language that should be used is VB (i.e., Visual Basic), among other things.

Once the first declaration has been retrieved and stored in a data structure, the process 700 continues with test operation 708 which determines whether there is another declaration in the file. This operation is similar to operation 702 described above. If there are no more declarations, flow branches NO to end step 714. If there is another declaration, flow branches YES to get operation 710, which gets the next declaration.

Following the retrieval of the next declaration, store operation 708 stores information related to the next declaration in a different data structure which is linked to the other data structures of the data model. As discussed above with respect to the first declaration, the information stored in the data structure may include not only the literal text of the next declaration but also information derived

from the literal text. As such, the data structure can be linked to the first data structure in a hierarchical manner if necessary.

Following the storing of the information in the next data structure, flow branches back to test operation 708 which determines whether the resource file, e.g., the ASP+ file, has any more declarations to read into the data model. If not, flow branches NO to the end of the process. However, if test operation 708 determines that there are more declarations, then flow branches YES to get operation 710 which gets the next declaration. Following get operation 710, store operation 708 stores information related to next declaration as described above. These steps 708, 710, and 712 continue until all declarations are retrieved in a sequential manner and stored in data structures. The data structures may be linked together in a hierarchical manner wherein nested objects, children, nodes or sub-elements of parent objects, are connected so that they are identifiable as sub-elements.

Create source code module 608 (FIG. 6) uses this data model to create the source code file of a particular source code language. It is important to note that the process of creating the source code file from the data model depends on the source code language. That is, a source code file written in a particular language typically has some specific format requirements, e.g., that all variable declarations precede the use of those variables. As such, since the compiler for that language is called, the proper program instructions must be placed in their proper locations within the source code file. In order to accomplish this task, the ASP+ file must be evaluated to determine what unique elements or features are in the file and the file must then be processed to produce the source code file. Since the source code file typically requires that a first set of elements or declarations must be at the beginning of the file, such as variable declarations, the entire data model should be analyzed for this information. Similarly since object information should be in the middle of the file, the entire data model should be analyzed for this information, during a second phase of the process, following the first phase of the process.

These phases are the functional process of determining the separate portions of code that should be written to the source code file (or intermediate data structure at operation 607), wherein the separate portions are logically combined based on resulting location within the source code file. As such, the phases may be considered "passes" or traversals of the singular data model and thus occur

consecutively. Alternatively however, the phases may occur as separate parallel processes, the results of the separate processes being merged into one source code file wherein the processes evaluate either the singular data model or separate copies of the data model. In yet another alternative, the functional processes or phases of determining and writing the separate portions of code may be performed during a single traversal, where the source code is selectively written to separate portions of the source code file or where the portions are written to separate files that are merged or linked to provide the proper connection between logical portions of source code. Therefore, although the separate analyses are described herein as being performed in a substantially consecutive manner, i.e., one after another, the analyses may be split and performed in a substantially simultaneous manner.

The operation or flow shown in FIG. 8 relates to an embodiment that generates source a code file relatively directly from the data model, i.e., omitting operation 607 shown in Fig. 6. The embodiment generates a source code file, such as a VB code file, where the variables must be declared in the beginning and the objects and methods follow the variable information. The create source code process 800 shown in FIG. 8 illustrates the separate phases as "passes" of the data model and as such, makes three passes through the data structures of the data model. In the first pass, the process is looking for variable and declaration information. In the second pass, the process is looking for object creation information, and in the third pass process 800 is looking for code rendering information.

The first pass begins as traversal operation 802 traverses the data model for variable and other declaration information. This step begins by analyzing the first data structure of the data model created by process 700. During the analysis of the first data structure, the process determines whether any specific lines of source code should be written to the source code file related to declarations.

Once the analysis of the first data structure has been completed, operation 804 generates and writes source code related to the variable and declaration information to the source code file. A writer object may be called to simply write the text given to it to the source code file. Thus, a call to the writer object includes both a parameter of the source code file name and a text string of the syntax for the particular programming language related to the variable or declaration information.

Alternatively, the text may be copied or otherwise written into the source code file using other known methods.

Operation 804 also must determine the proper syntax to deliver to the writer object. In essence, if the current data structure represents a server side control, operation 804 generates source code for it. Otherwise 804 copies it to the source code file. In essence, Operation 804 may do one of three things. First, generate operation 804 may determine that the information in the ASP+ file is literal text and as such, should be written directly to the source code file. This may be the case where an HTML tag is inserted in the ASP+ file and in this case, no source code declaration is generated, instead the information is simply copied to the source file. Second, the operation 804 may determine that the text requires a simple translation, a one to one association with a proper syntax form for the specific source code language. Operation 804, in such a case, will either look up the one-to-one translation or, if hard-coded in the page factory module, the one-to-one translation is performed and the resulting information is supplied to the writer object. Third, if a one-to-one translation is not available, the operation 804 may determine that a more complex translation is required and as such may call a module or subsequent look-up table operation to perform the operation of creating the proper source code syntax to be conducted to the writer object. Typically, for declaration information the translation is straightforward.

Once operation 804 generates and writes source code for the first data structure, test operation 806 detects whether there are any more data structures in the data model to be analyzed. If there are more data structures to be analyzed during this first pass of the data model, flow branches YES to the beginning of process 800 in that the transversal is made at 802 to the next data structure. In such a case, the next data structure will be analyzed as discussed above to generate source code syntax and to call the writer object for writing that syntax code into the source code file.

If test operation 806 determines that there are no more data structures in the data model to be analyzed during this first pass, flow branches NO to traverse operation 808, and the second pass through data structures begins. Traverse operation 808 begins the next phase of writing codes typically to write the object

building and methods creation. Traverse operation 808 begins at the top of the data model and returns the data structure at the beginning of the data model.

Next, generate and write source code operation 810 analyzes the first data structure to determine if information related to object creation is located in that first data structure. If such object creation information exists in that first data structure then syntax for the specific programming language is created, generated and sent to a writer object to write to the source code file. In essence, operation 810 is similar to operation 804 discussed above wherein the data structure is analyzed for a specific type of information and once that information is gleaned from the data structure, the operation performs some sort of translation to create source code language related to that information. If the information is literal, then operation 810 can transfer that information directly into the source code file. Similarly if that information requires a simple translation then that simple translation may be hard coded into the program or may be part of a look up table type operation. Additionally, if a more complex structure is required then a look up table or other module can be called to process the information and to create the proper syntax source code for the source code file.

Following operation 810, query step 812 tests whether there are any more data structure in the data model to be analyzed. If there are more data structures in the data model to be analyzed for object creation, then flow branches is YES to traverse operation 808 which traverses to the next data structure. Thus, operation 808 returns the next data structure and operation 810 analyzes that data structure for information related to object creation. This object creation information, as discussed above translates into source code lines that can be sent to the writer object and appended to the source code file. Steps 812, 808 and 810 repeat until all source code has been written with respect to object creation.

If determination step 812 determines that there are no more data structures in the data model to be analyzed for object creation then flow branches NO to operation 814, and the second pass is completed. The third pass begins with operation 814 at the top of the data model. As discussed above with respect to operations 808 and 802 this traversal step initially retrieves the first data structure in the data model. Following the retrieval of the first data structure in the data model at operation 814, operation 816 analyzes that data structure for code rendering

information. Code rendering information translates into specific syntax in the programming language to be sent to the writer and appended to the source code file. Code rendering methods may be located towards the end of the source code file, and therefore a third pass is implemented following the object creation pass. Alternatively, the cod rendering methods may be generated during a simultaneous thread process and appended to the end of the source file.

Operation 816 is quite similar to operations 810 and 804 discussed above wherein this source code information can be gleaned from the code rendering information found in the data structure. Moreover, as discussed above, the information may be inserted directly, it may be translated one-to-one ratio, or it may be required that a more complex module is used to generate the syntax for the source code file. Following operation 816, detect operation 818 determines whether there are any more data structures in the data model.

If operation 818 detects that there are more data structures to be analyzed during this third pass, flow branches YES back to 814 for traversal to the next data structure. Once the next data structure has been called then operation 818 generates and writes source code relating to code rendering information in that next data structure.

If operation 818 determines that there are no more data structures to be analyzed then flow branches NO to the end of the process at operation connection 820.

As can be seen by the above description, several passes are made through the data model. This is due to the fact that each declaration in the data model may require specific lines of code that should be placed in various spots in the source code file. However, in writing the source code file the writer object can only write sequentially, continuously appending to the file and cannot insert lines of source code in between previously written lines of code. Therefore, the data model must be traversed more than once to glean the proper information that belongs in the first or upper portion of the source code, in the second or middle portion of the source code, and a third or last portion of the source code file. It is foreseen that the number of passes may vary depending on the number of portions of code that exist for the various computer languages.

The operation or flow shown in FIG. 9 relates to an alternative embodiment that generates an intermediate data structure from the data model and then generates the source code file from the intermediate data structure, i.e., operations 607 and 608 shown in FIG. 6. The flow 900 of this embodiment is similar to the flow shown in FIG. 8 except that operations 804, 810 and 816 generate source code of a particular language and operations 904, 910 and 916 generate portions of the intermediate data structure, instead of generating source code. Thus, operations 904, 910 and 916 perform operations similar to operations 804, 810 and 816 in generating information related to information in the data model. However, the generated information is generic and may be used at a later time to create source code files of different source code languages.

Once operation has determined that there are no more data structures in the data model to be analyzed then the intermediate data structure is complete and flow branches to create operation 920. Create operation creates a source code file from the intermediate data structure. Since the data structure is a generic description of the source code file, operation 920 translates the intermediate data structure from a generic description to a specific language code file.

FIG. 10 is an example of a web page resource file, i.e., an ASP+ file authored by a web page developer, the file being the subject of the process 600 to generate a source code file. The ASP+ page shown in FIG 10 has a directives line at line 1; a server side script block at lines 3 through 13; and a server side control declaration block at lines 16 through 21. The directives line is used by the process 600 to determine what type of source code language to use and to give it a general description but otherwise doesn't result in any code in the source code file. All code written within a code declaration block "<script runat=server> </script>" is conceptually treated as page member declarations (variables, properties, methods) and is directly inserted into the source file of the generated file as shown in FIG. 11.

FIG. 11 is an exemplary source code file that has been created by the process of creating a source code using the ASP+ file shown in FIG 10. The first line of FIG 11 shows that the page class that is going to be created by the source code file, when compiled, inherits from System.ASP.WebForms.Page. Inheriting from this class is an important step as it provides many of the control functions for page class. As a default, the generated source-file subclasses the "System.ASP.WebForms.Page"

base class. Developers can optionally specify an alternative class using the "Inherits" attribute provided on the directive line.

Lines 3 and 4 show the variable declarations that were created during the first pass through the data model.

During the second pass the information shown in line 6 was created to create a new control object "DataList" and name it MyData. In an embodiment, all code written within in the ASP+ file declaration block "<script runat=server> </script>" is conceptually treated as page member declarations (variables, properties, methods) and is directly inserted into the source file of the generated file as discussed above. Thus, the lines of code 8-13 were directly inserted as literal text. During the creation of the data model 606, information related to literal text was stored in a string or an array, essentially above or apart from the data model, such that it was not traversed during the multiple passes (FIG. 8). During the creation of the data model, it was determined that this information would be inserted directly, thus enabling its storage as an array having a reference pointer in the data model. During the generation of the code, this information is simply inserted into the source code file.

Lines 15 to 39 represent a section of code that was written during the second pass related to building control object information. Lines 15-20 represent code used in the creation of a top-level object. The top-level object is a container type object for the entire page, such as the page object 314 shown in FIG. 3. This control object is built relatively independently of the of the substantive code shown in FIG. 10, as all ASP+ pages created will have this type of control object.

Lines 22-27 of FIG. 11 represent code used in the creation of a child control object named MyList. As shown in line 16 of FIG. 10, a tablelist control was defined and given an id "MyList" The information from FIG. 10, line 16 translates at step 810 (FIG. 8) into the lines 22-27 shown in FIG. 11. In essence, when operation 810 parses the code portion on line 16 of FIG. 10, a recognition occurs that a control of type "tablelist" should be created. Once recognized, the basic information in lines 22-27 shown in Fig. 11 are either looked up from a table or generated using known calculations while inserting the proper variables such as "MyList" to generate the code shown in the lines 22-27. Since the system is designed to recognize this type of controls, the code shown in FIG. 11 can be generated. Once generated, operation

810 also writes the code to the file.

Similarly, lines 29-34 of Fig. 11 represent another control object, in this case an object of the type "template." Templates are special objects in that they are also container controls, and in that they actually generated at run time. In any event, the code required to generate the control object for the template is generated at step 810 (following the test 812 which determined that there were more data structures in the model.) As discussed above, a recognition occurs that a template control object should be generated and therefore the code shown in FIG. 11 is generated and written to the file. Importantly, the code for the template requires information related to its child controls, evidenced by the references to "control3" in lines 32 and 33 in FIG. 11. Therefore, when the data model was created at 606 (FIG. 6), this information or an indication to determine this information was stored with the template control information. Again, the code is predetermined for template control objects such that the generation of the code is straightforward.

A child control is created within the template using the code on line 19 of FIG. 10 and its corresponding translated code at lines 36-39 of FIG. 11. Line 19 of FIG. 10 calls for a control of type "label" having an id "MyLabel". As with the tablelist control and the template control, the label control is a known type of control that operation 810 recognizes and is able to then generate the proper code and write it to the file as shown in lines 36-39 of Fig. 11.

Lines 41 through 51 were created, generated and written to the source code file last during the last pass of the data model. The lines of code at lines 41-51 represent render methods which are called to render the HTML code that becomes part of the response to the client. The rendering code is generated based on a recognition, during step 816 that such code should be generated. Once recognized, the code is simply determined from a lookup table or generated as needed.

The code shown in FIG. 11 represents the server-side generated code from a particular dynamic web page content file, i.e., the file shown in FIG. 10. Once the file shown in FIG. 11 is completed, the file may be compiled as discussed above with respect to operation 610 shown in FIG. 6. The compilation results in a class that can be used to generate the controls of the dynamic web page content file. The class is stored in cache or other memory and may be used as many times as desired to instantiate the objects for the page. Both the compilation and persistence of the

resulting class may be done "in-memory" such that there is no need to access the disk, which is generally slower than doing things in memory.

All operations necessary to setup, activate, and run the server-side page are encapsulated within the dynamically compiled page class. As a result, no additional configuration/parsing of files is necessary during page setup. Moreover, the original ".aspx" or ASP+ file is not touched again, and a "runtime host environment," e.g., an ASP script engine is not required during page execution.

The embodiments of the invention described herein are implemented as logical steps in one or more computer systems. The logical operations of the present invention are implemented (1) as a sequence of processor-implemented steps executing in one or more computer systems and (2) as interconnected machine modules within one or more computer systems. The implementation is a matter of choice, dependent on the performance requirements of the computer system implementing the invention. Accordingly, the logical operations making up the embodiments of the invention described herein are referred to variously as operations, steps, objects, or modules.

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

4 Brief Description of Drawings

FIG. 1 illustrates a web server for dynamically generating web page content for display on a client in an embodiment of the present invention.

FIG. 2 illustrates a flow diagram of operations for processing and rendering client-side user interface elements using server-side control objects in an embodiment of the present invention.

FIG. 3 illustrates exemplary modules in a web server used in an embodiment of the present invention.

FIG. 4 illustrates an exemplary dynamic content file (e.g., an ASP+ page) in an embodiment of the present invention.

FIG. 5 illustrates an exemplary system useful for implementing an embodiment of the present invention.

FIG. 6 illustrates a process flow diagram representing processing of a page object in an embodiment of the present invention.

FIG. 7 illustrates a process flow diagram representing the parsing of a seven-side application to create a data model.

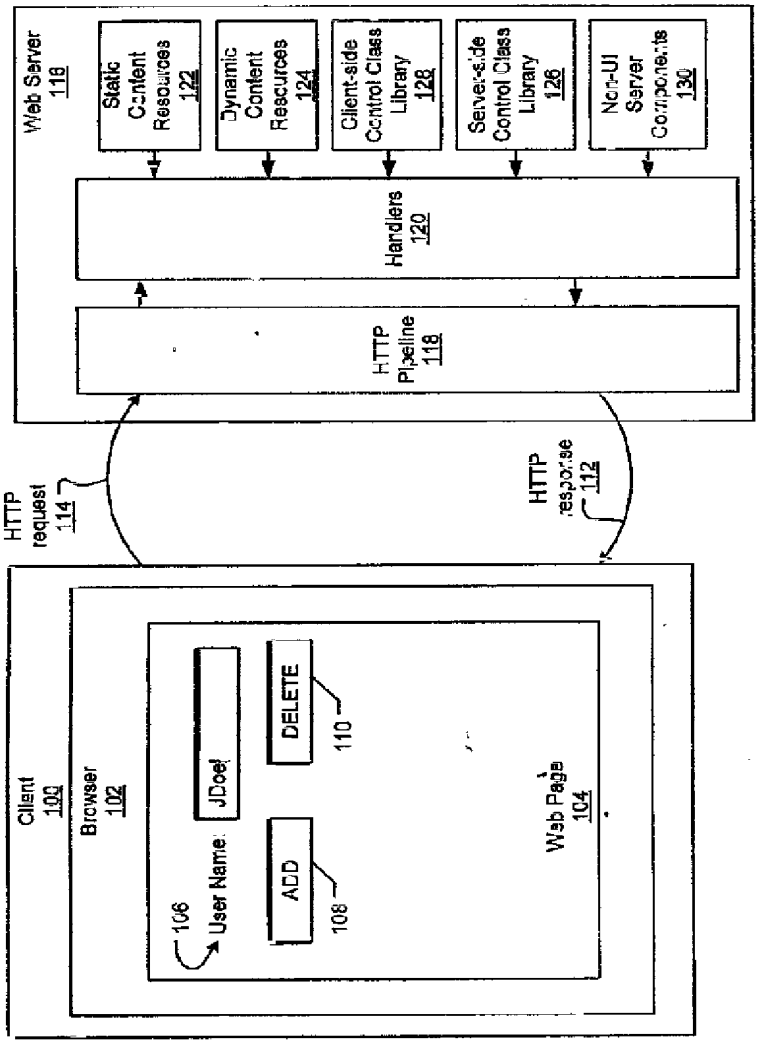
FIG. 8 illustrates a process flow diagram representing the traversal of the data model to create a source code file.

FIG. 9 illustrates a process flow diagram representing the traversal of the data model to create a source code file in an alternative embodiment of the present invention.

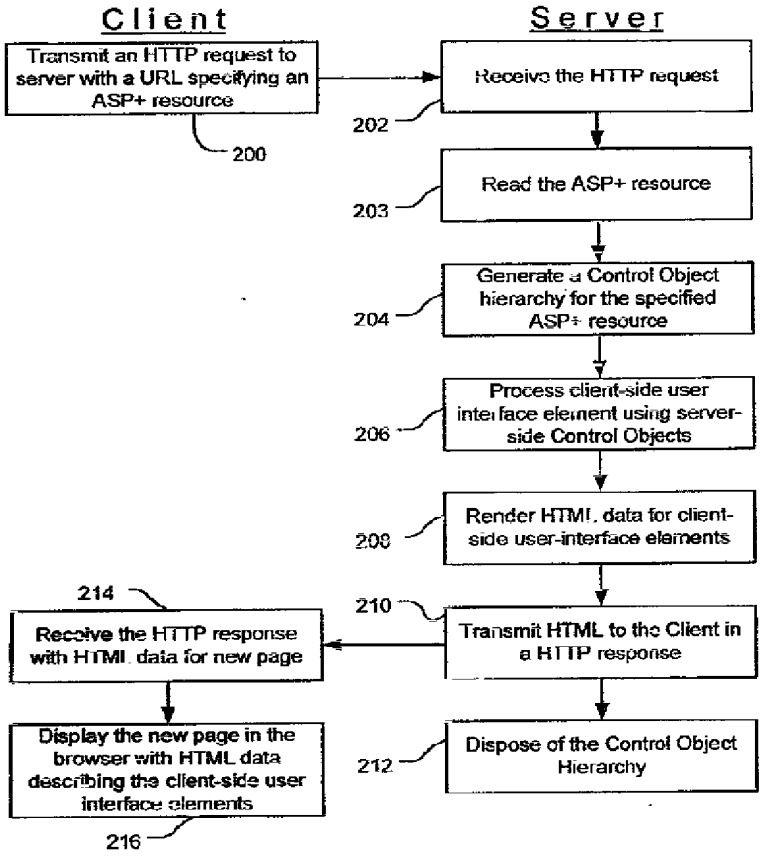
FIG. 10 illustrates an example of an ASP+ page that can be parsed in accordance with the present invention.

FIG. 11 illustrates an example of a source code file created using the present invention wherein the initial ASP+ file shown in FIG. 10.

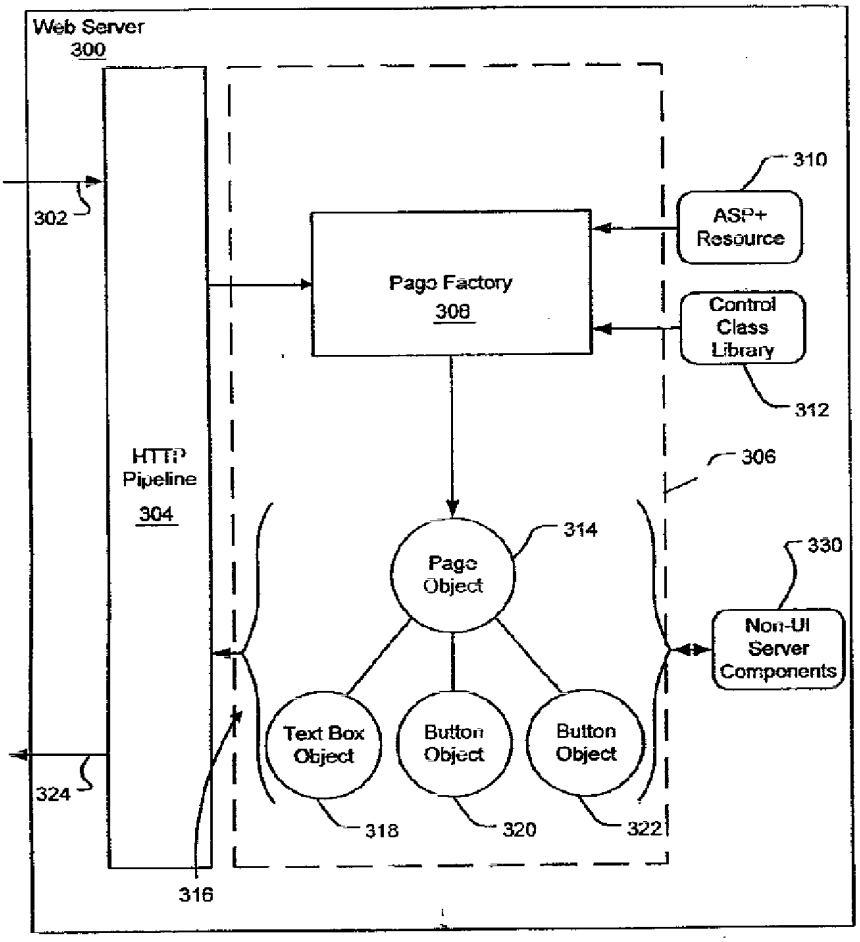
【図1】


Web Server
118
Static Content Resources
122
Dynamic Content Resources
124
Client-side Control Class Library
128
Server-side Control Class Library
126
Non-UI Server Components
130
Handlers
120
HTTP Pipeline
118
HTTP request
114
HTTP response
112
Client
100
Browser
102
JDoe
DELETE
110
User Name:
106
ADD
108
Web Page
104

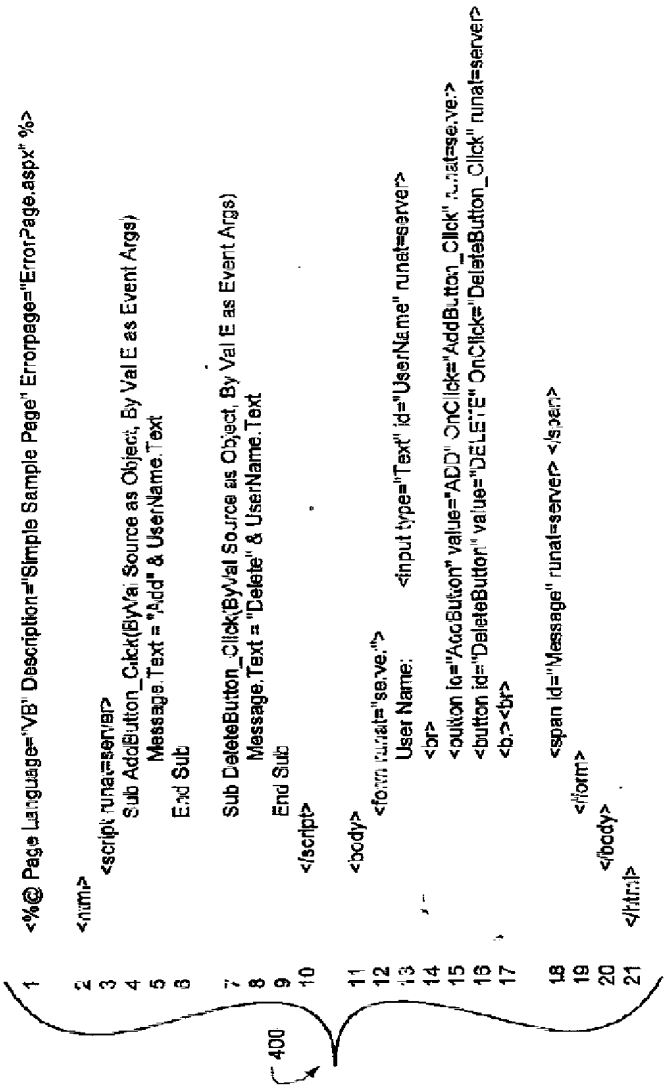
【図2】


Client
Server
Transmit an HTTP request to server with a URL specifying an ASP+ resource
200
Receive the HTTP request
202
Read the ASP+ resource
203
Generate a Control Object hierarchy for the specified ASP+ resource
204
Process client-side user interface element using server-side Control Objects
206
Render HTML data for client-side user-interface elements
208
Transmit HTML to the Client in a HTTP response
210
Dispose of the Control Object Hierarchy
212
Receive the HTTP response with HTML data for new page
214
Display the new page in the browser with HTML data describing the client-side user interface elements
216

【図３】


Web Server
300
302
HTTP
Pipeline
304
Page Factory
308
ASP+
Resource
310
Control
Class
Library
312
306
Page
Object
314
330
Non-UI
Server
Components
Text Box
Object
Button
Object
Button
Object
318
320
322
324
316

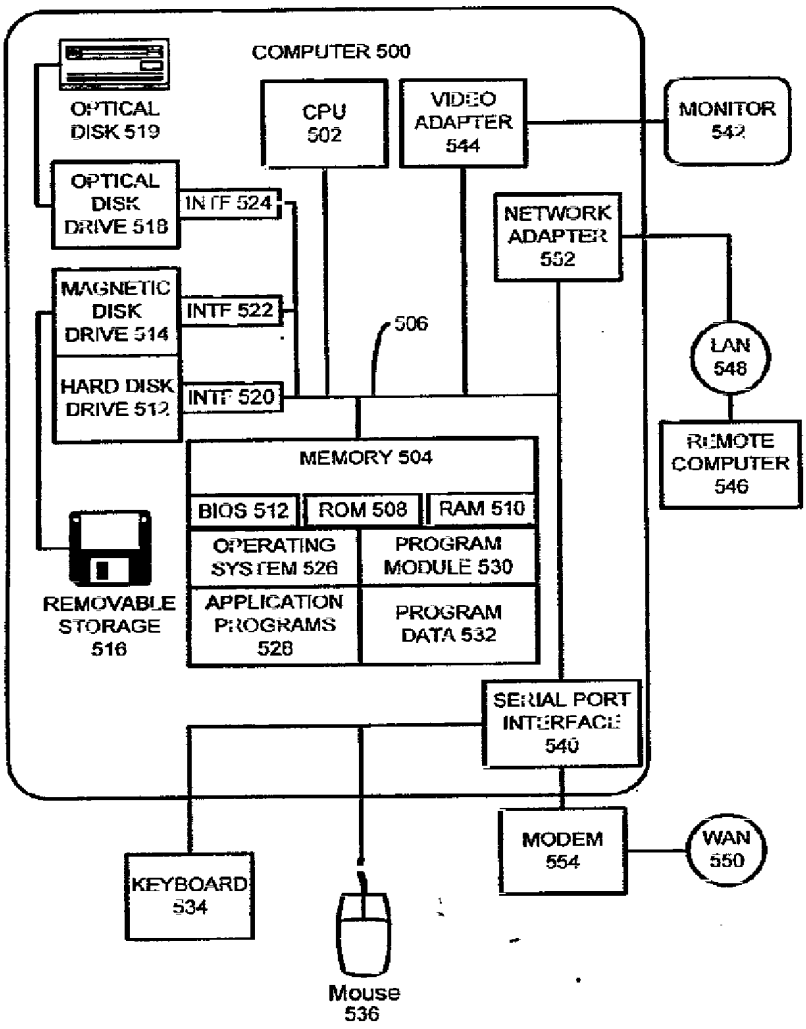
【図4】


400
<%@ Page Language="VB" Description="Simple Sample Page" Errorpage="ErrorPage.aspx" %>
<html>
<script runat=server>
Sub AddButton_Click(ByVal Source as Object, By Val E as Event Args)
Message.Text = "Add" & UserName.Text
End Sub
Sub DeleteButton_Click(ByVal Source as Object, By Val E as Event Args)
Message.Text = "Delete" & UserName.Text
End Sub
</script>
<body>
<form runat="server">
User Name:
<input type="Text" id="UserName" runat=server>

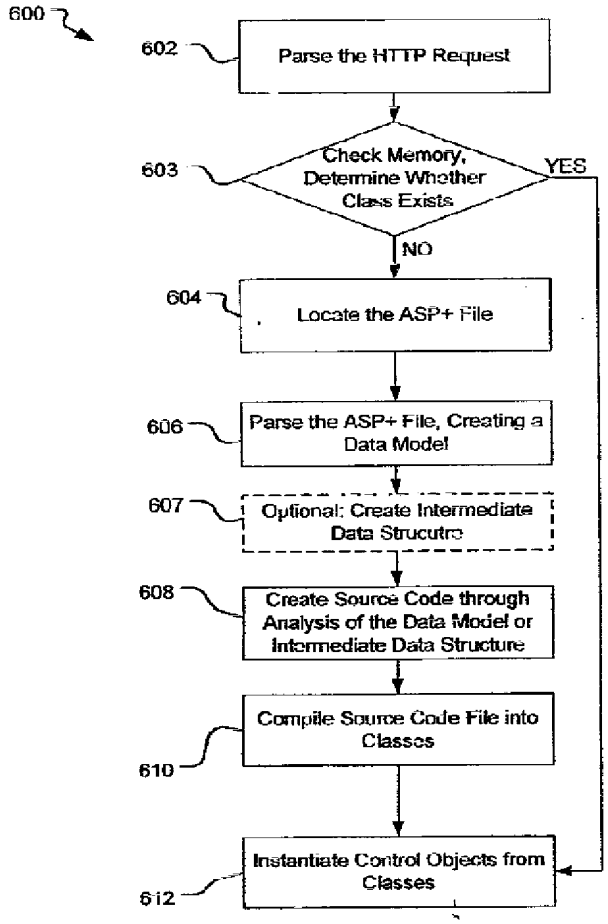
<button id="AddButton" value="ADD" OnClick="AddButton_Click" runat=server>
<button id="DeleteButton" value="DELETE" OnClick="DeleteButton_Click" runat=server>
<b>

</form>
</body>
</html>

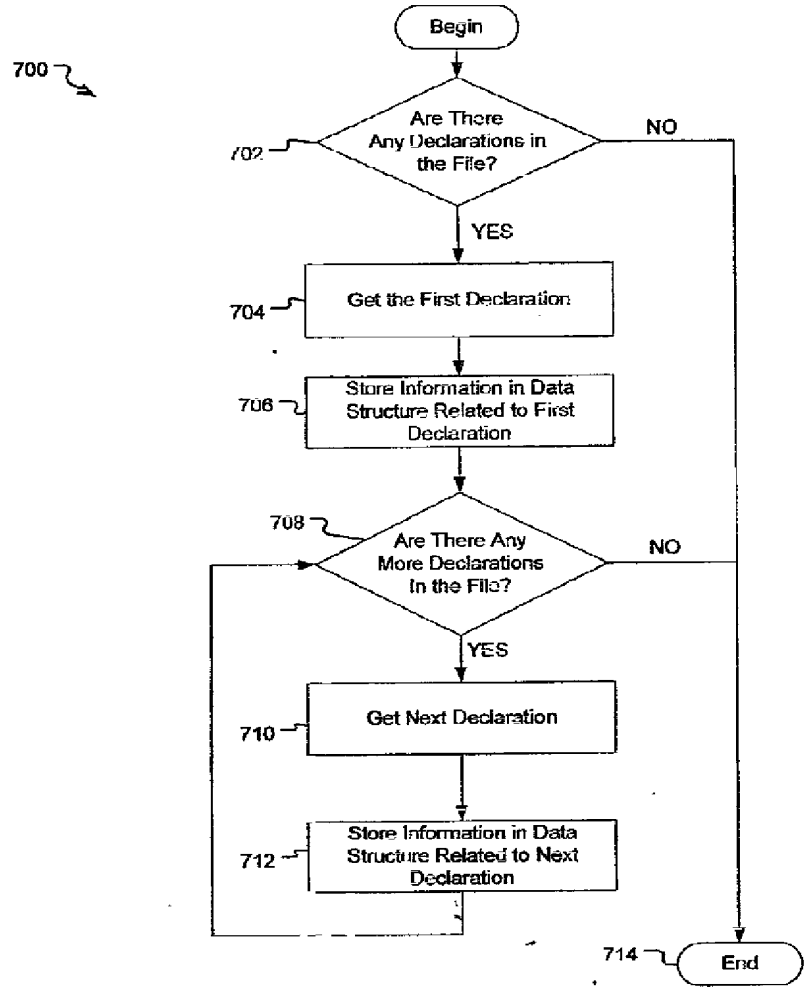
【図5】


COMPUTER 500
OPTICAL DISK 519
CPU 502
VIDEO ADAPTER 544
MONITOR 542
OPTICAL DISK DRIVE 518
INTF 524
NETWORK ADAPTER 552
MAGNETIC DISK DRIVE 514
INTF 522
506
LAN 548
HARD DISK DRIVE 512
INTF 520
REMOTE COMPUTER 546
MEMORY 504
BIOS 512
ROM 508
RAM 510
OPERATING SYSTEM 526
PROGRAM MODULE 530
REMOVABLE STORAGE 516
APPLICATION PROGRAMS 528
PROGRAM DATA 532
SERIAL PORT INTERFACE 540
MODEM 554
WAN 550
KEYBOARD 534
Mouse 536

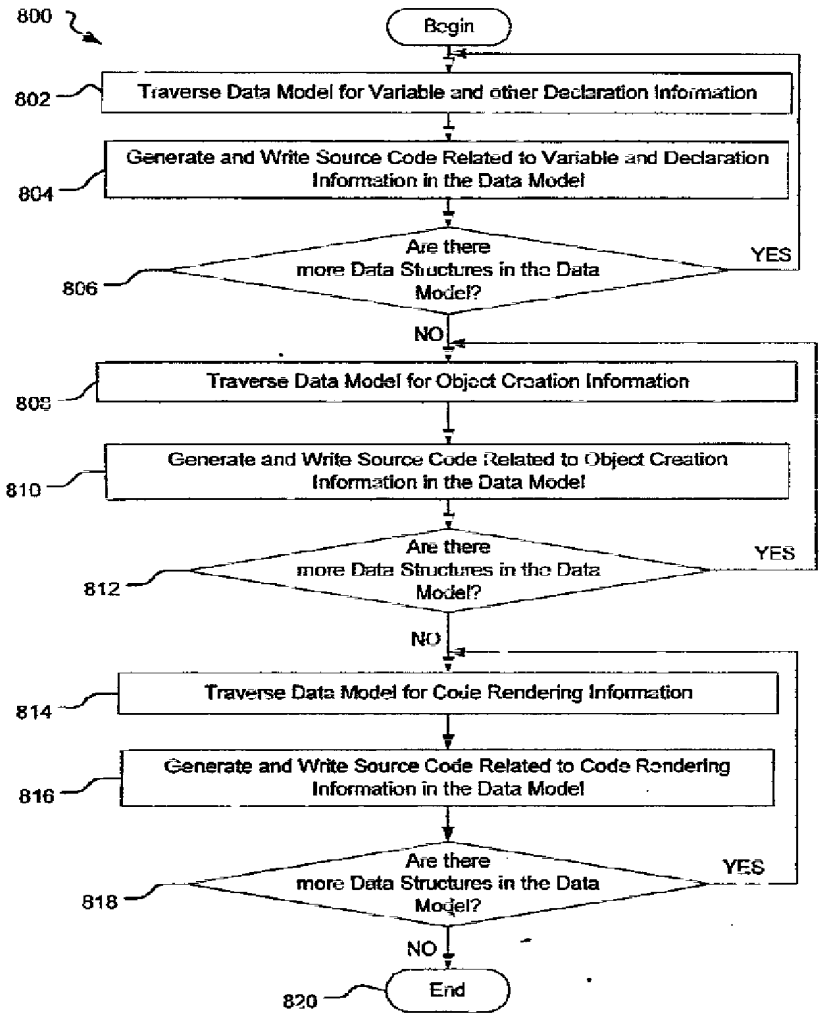
【図6】


600
602
Parse the HTTP Request
603
Check Memory, Determine Whether Class Exists
YES
NO
604
Locate the ASP+ File
606
Parse the ASP+ File, Creating a Data Model
607
Optional: Create Intermediate Data Strucutre
608
Create Source Code through Analysis of the Data Model or Intermediate Data Structure
610
Compile Source Code File into Classes
612
Instantiate Control Objects from Classes

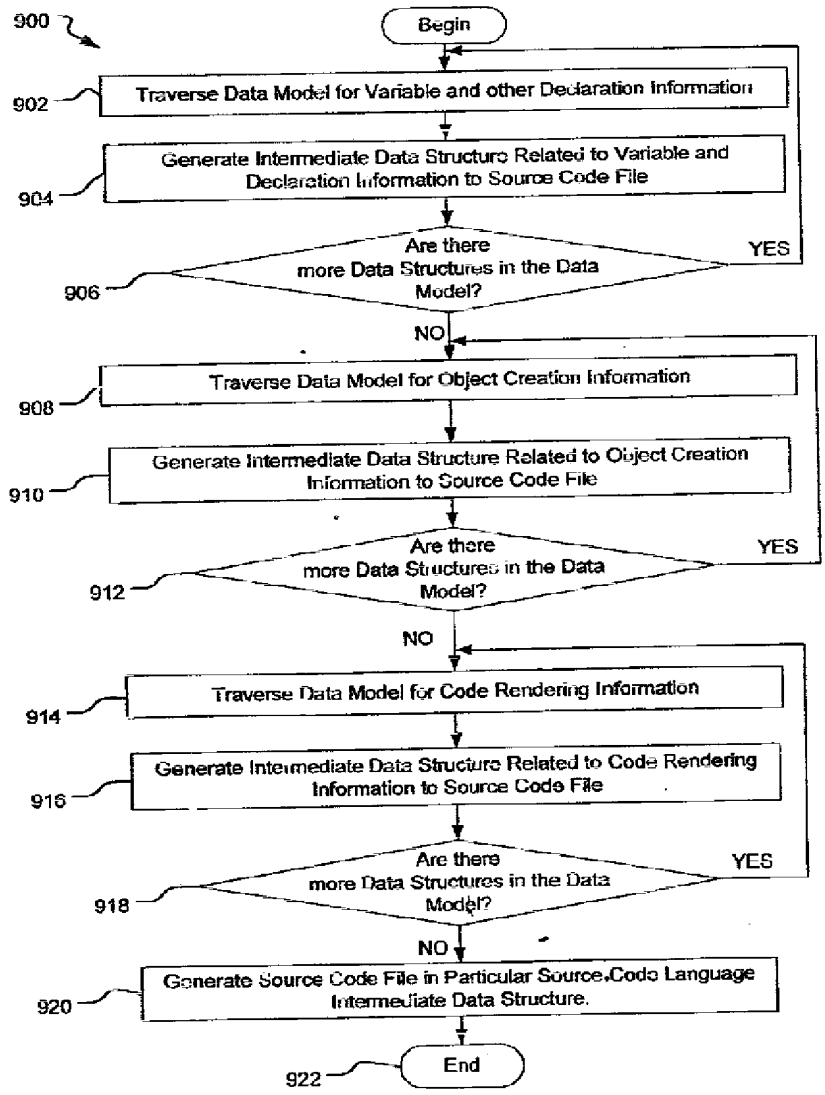
【図7】


700
Begin
Are There Any Declarations in the File?
702
NO
YES
Get the First Declaration
704
Store Information in Data Structure Related to First Declaration
706
Are There Any More Declarations In the File?
708
NO
YES
Get Next Declaration
710
Store Information in Data Structure Related to Next Declaration
712
End
714

【図8】


800
Begin
802
Traverse Data Model for Variable and other Declaration Information
804
Generate and Write Source Code Related to Variable and Declaration Information in the Data Model
806
Are there more Data Structures in the Data Model?
YES
NO
808
Traverse Data Model for Object Creation Information
810
Generate and Write Source Code Related to Object Creation Information in the Data Model
812
Are there more Data Structures in the Data Model?
YES
NO
814
Traverse Data Model for Code Rendering Information
816
Generate and Write Source Code Related to Code Rendering Information in the Data Model
818
Are there more Data Structures in the Data Model?
YES
NO
820
End

【図9】


900
Begin
902
Traverse Data Model for Variable and other Declaration Information
904
Generate Intermediate Data Structure Related to Variable and Declaration Information to Source Code File
906
Are there more Data Structures in the Data Model?
YES
NO
908
Traverse Data Model for Object Creation Information
910
Generate Intermediate Data Structure Related to Object Creation Information to Source Code File
912
Are there more Data Structures in the Data Model?
YES
NO
914
Traverse Data Model for Code Rendering Information
916
Generate Intermediate Data Structure Related to Code Rendering Information to Source Code File
918
Are there more Data Structures in the Data Model?
YES
NO
920
Generate Source Code File in Particular Source Code Language Intermediate Data Structure.
922
End

【図10】

```
1   <%@ language="VB" Description="Test of the TableList control" %>
2
3   <script runat=server> .
4     public MyData as New DataList
5
6     Overrides Sub Init()
7       MyData.Add "Name1"
8       MyData.Add "Name2"
9       MyData.Add "Name3"
10
11      Set MyList.DataSource = MyData
12    End Sub
13  </script>
14
15  <%%>
16  <wfc:TableList id="MyList" runat=server>
17    <template:ItemTemplate runat=server>
18  <%%>
19        <wfc:Label id="MyLabel" databinding="text:DataItem" runat=server/>
20    </template:ItemTemplate>
21  </wfc:TableList>
```

【図11】

```
1   Inherits System.ASP.WebForms.Page
2
3   Dim MyList as TableList
4   Dim __control3 as Label
5
6   public MyData as New DataList
7
8   Overrides Sub Init()
9     MyData.Add "Name1"
10    MyData.Add "Name2"
11    MyData.Add "Name3"
12    Set MyList.DataSource = MyData
13  End Sub
14
15  Public Sub _tmp_aspx0__BuildControl__control0(ByVal __ctrl as ContainerControl)
16    __ctrl.SetRenderMethodDelegate New RenderMethod(AddressOf
17  Me._tmp_aspx0__Render__control0)
18    _tmp_aspx0__BuildControlMyList
19    __ctrl.AddParsedSubControl MyList
20  End Sub
21
22  Public Sub _tmp_aspx0__BuildControlMyList
23    set MyList = new TableList
24    MyList.ID = "MyList"
25    set MyList.ItemTemplate = new CompiledTemplateBuilder(new
26  BuildTemplateMethod(AddressOf me._tmp_aspx0__BuildControl__control2))
27  End Sub
28
29  Public Sub _tmp_aspx0__BuildControl__control2(ByVal __ctrl as ContainerControl)
30    __ctrl.SetRenderMethodDelegate New RenderMethod(AddressOf
31  Me._tmp_aspx0__Render__control2)
32    _tmp_aspx0__BuildControl__control3
33    __ctrl.AddParsedSubControl __control3
34  End Sub
35
36  Public Sub _tmp_aspx0__BuildControl__control3
37    set __control3 = new Label
38    __control3.ID = "MyLabel"
39  End Sub
40
41  Public Sub _tmp_aspx0__Render__control0(ByVal output as HtmlTextWriter,
42  ByVal __container as ContainerControl)
43    Call __container.Controls(0).RenderControl(output)
44    output.Write(""&vbCRLF &""&vbCRLF &"")
45  End Sub
46
47  Public Sub _tmp_aspx0__Render__control2(ByVal output as HtmlTextWriter,
48  ByVal __container as ContainerControl)
49    Call __container.Controls(0).RenderControl(output)
50    output.Write(""&vbCRLF &"  ")
51  End Sub
```

A method and apparatus to create an intermediate language or source code file from a server-side resource or dynamic web page file. The source code can then be compiled into an executable class allowing for rapid generation of web page control objects that perform server-side functions, including the rendering of client responses. The code generation scheme of the present invention is capable of creating control objects connected in a hierarchy to handle event processing and the setting of attributes to the specific objects.

2 Representative Drawing

Fig. 6